34th International Conference on Foundation of Software Technology and Theoretical Computer Science

FSTTCS 2014, December 15–17, 2014, New Delhi, India

Edited by

Venkatesh Raman
S. P. Suresh

Editors

Venkatesh Raman
The Institute of Mathematical Sciences
Chennai 600113
India
`vraman@imsc.res.in`

S. P. Suresh
Chennai Mathematical Institute
Chennai 603103
India
`spsuresh@cmi.ac.in`

ACM Classification 1998
D.2.4 Software/Program Verification, F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.3 Formal Languages

ISBN 978-3-939897-77-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-77-4.

Publication date
December, 2014

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2014.i

ISSN 1868-8969

www.dagstuhl.de/lipics

Contents

Preface ix

Conference Organization xi

External Reviewers xiii

Invited Talks

New Developments in Iterated Rounding
Nikhil Bansal . . . 1

Reasoning about distributed systems: WYSIWYG
C. Aiswarya and Paul Gastin . . . 11

Colour Refinement: A Simple Partitioning Algorithm with Applications From Graph Isomorphism Testing to Machine Learning
Martin Grohe . . . 31

Properties and Utilization of Capacitated Automata
Orna Kupferman and Tami Tamir . . . 33

Algorithms, Games, and Evolution
Erick Chastain, Adi Livnat, Christos H. Papadimitriou, and Umesh V. Vazirani . 45

The Polynomial Method in Circuit Complexity Applied to Algorithm Design
Richard Ryan Williams . . . 47

Contributed Papers

Session 1A

Vertex Exponential Algorithms for Connected f-Factors
Geevarghese Philip and M. S. Ramanujan . . . 61

Connecting Vertices by Independent Trees
Manu Basavaraju, Fedor V. Fomin, Petr A. Golovach, and Saket Saurabh . . . 73

Tree Deletion Set Has a Polynomial Kernel (but no OPT$^{\mathcal{O}(1)}$ Approximation)
Archontia C. Giannopoulou . . . 85

Editing to Eulerian Graphs
Konrad K. Dabrowski, Petr A. Golovach, Pim van 't Hof, and Daniël Paulusma . 97

Parameterized Complexity of Fixed-Variable Logics
Christoph Berkholz and Michael Elberfeld . . . 109

Session 1B

Synchronizing Words for Weighted and Timed Automata
Laurent Doyen, Line Juhl, Kim G. Larsen, Nicolas Markey, and Mahsa Shirmohammadi . . . 121

Finite-Valued Weighted Automata
Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin 133

First-order Definable String Transformations
Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi 147

Regular Sensing
Shaull Almagor, Denis Kuperberg, and Orna Kupferman 161

Symbolic Solving of Extended Regular Expression Inequalities
Matthias Keil and Peter Thiemann 175

Session 2A

Solving the Stable Set Problem in Terms of the Odd Cycle Packing Number
Adrian Bock, Yuri Faenza, Carsten Moldenhauer, and Andres Jacinto Ruiz-Vargas 187

Lift & Project Systems Performing on the Partial Vertex Cover Polytope
Konstantions Georgiou and Edward Lee 199

Replica Placement on Directed Acyclic Graphs
Sonika Arora, Venkatesan T. Chakaravarthy, Kanika Gupta, Neelima Gupta, and Yogish Sabharwal 213

Maintaining Approximate Maximum Matching in an Incremental Bipartite Graph in Polylogarithmic Update Time
Manoj Gupta 227

Session 2B

The Complexity of Counting Models of Linear-time Temporal Logic
Hazem Torfah and Martin Zimmermann 241

Extending Temporal Logics with Data Variable Quantifications
Fu Song and Zhilin Wu 253

Generalized Data Automata and Fixpoint Logic
Thomas Colcombet and Amaldev Manuel 267

Consistency of Injective Tree Patterns
Claire David, Nadime Francis, and Filip Murlak 279

Session 3A

Asymptotically Optimal Encodings for Range Selection
Gonzalo Navarro, Rajeev Raman, and Srinivasa Rao Satti 291

Output-Sensitive Pattern Extraction in Sequences
Roberto Grossi, Giulia Menconi, Nadia Pisanti, Roberto Trani, and Søren Vind .. 303

Robust Proximity Search for Balls Using Sublinear Space
Sariel Har-Peled and Nirman Kumar 315

The Benes Network is q(q−1)/2n-Almost q-set-wise Independent
Efraim Gelman and Amnon Ta-Shma 327

Notes on Counting with Finite Machines
Dmitry Chistikov 339

Session 3B

Mixed Nash Equilibria in Concurrent Terminal-Reward Games
Patricia Bouyer, Nicolas Markey, and Daniel Stan 351

Quantitative Games with Interval Objectives
Paul Hunter and Jean-François Raskin 365

Playing Safe
Thomas Colcombet, Nathanaël Fijalkow, and Florian Horn 379

Metaconfluence of Calculi with Explicit Substitutions at a Distance
Flávio L. C. de Moura, Delia Kesner, and Mauricio Ayala-Rincón 391

Behavioral Metrics via Functor Lifting
Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König 403

Session 4

Foundation of Diagnosis and Predictability in Probabilistic Systems
Nathalie Bertrand, Serge Haddad, and Engel Lefaucheux 417

Lipschitz Robustness of Finite-state Transducers
Thomas A. Henzinger, Jan Otop and Roopsha Samanta 431

Session 5A

Separating Cook Completeness from Karp-Levin Completeness Under a Worst-Case Hardness Hypothesis
Debasis Mandal, A. Pavan, and Rajeswari Venugopalan 445

Constructing Small Tree Grammars and Small Circuits for Formulas
Danny Hucke, Markus Lohrey, and Eric Noeth 457

One Time-traveling Bit is as Good as Logarithmically Many
Ryan O'Donnell and A. C. Cem Say 469

New Bounds for the Garden-Hose Model
Hartmut Klauck and Supartha Podder 481

Homomorphism Polynomials Complete for VP
Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh 493

Session 5B

Computing Information Flow Using Symbolic Model-Checking
Rohit Chadha, Umang Mathur, and Stefan Schwoon 505

Information Leakage of Non-Terminating Processes
Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, Pasquale Malacaria, and Andrzej Wąsowski 517

Multiple-Environment Markov Decision Processes
Jean-François Raskin and Ocan Sankur 531

Summary-Based Inter-Procedural Analysis via Modular Trace Refinement
Franck Cassez, Christian Müller, and Karla Burnett 545

A Two-Level Logic Approach to Reasoning about Typed Specification Languages
Mary Southern and Kaustuv Chaudhuri 557

Session 6A

On the Complexity of Computing Maximum Entropy for Markovian Models
Taolue Chen and Tingting Han 571

New Time-Space Upperbounds for Directed Reachability in High-genus and H-minor-free Graphs
Diptarka Chakraborty, A. Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin Forrest Yang 585

Polynomial Min/Max-weighted Reachability is in Unambiguous Log-space
Anant Dhayal, Jayalal Sarma, and Saurabh Sawlani 597

Session 6B

On Bounded Reachability Analysis of Shared Memory Systems
Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan 611

Parameterized Communicating Automata: Complementation and Model Checking
Benedikt Bollig, Paul Gastin, and Akshay Kumar 625

Distributed Synthesis for Acyclic Architectures
Anca Muscholl and Igor Walukiewicz 639

Verification of Dynamic Register Automata
Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmet Kara, and Othmane Rezine .. 653

Preface

The 34th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014) was held at the India International Centre, New Delhi, from December 15 to December 17, 2014.

The program consisted of 6 invited talks and 47 contributed papers. This proceedings volume contains the contributed papers and abstracts of invited talks presented at the conference. The proceedings of FSTTCS 2014 is published as a volume in the LIPIcs series under a Creative Commons license, with free online access to all, and with authors retaining rights over their contributions.

The 47 contributed papers were selected from a total of 162 submissions. We thank the program committee for its efforts in carefully evaluating and making these selections. We thank all those who submitted their papers to FSTTCS 2014. We also thank the external reviewers for sending their informative and timely reviews.

We are particularly grateful to the invited speakers: Nikhil Bansal (TU Eindhoven), Paul Gastin (LSV, ENS Cachan), Martin Grohe (RWTH Aachen), Orna Kupferman (Hebrew University), Umesh Vazirani (UC Berkeley) and Ryan Williams (Stanford University), who readily accepted our invitation to speak at the conference.

There was one pre-conference workshop, New Developments in Exact Algorithms and Lower Bounds, and two post-conference workshops, INFINITY 2014 and Recent Advances in Cryptography, all held at IIT Delhi. We thank the organisers of the workshops.

On the administrative side, we thank the organizing committee led by Prof. Ragesh Jaiswal (IIT Delhi), who put in many months of effort in ensuring excellent conference arrangements at India International Centre and arrangements for the workshops at the IIT Delhi campus. We also thank the Easychair team whose software has made it very convenient to do many conference related tasks. Finally, we thank the Dagstuhl LIPIcs staff for their coordination in the production of this proceedings, particularly Marc Herbstritt who was very prompt and helpful in answering our questions.

Venkatesh Raman and S. P. Suresh
December 2014

Conference Organization

Programme Chairs

Venkatesh Raman(IMSc, Chennai)
S. P. Suresh (CMI, Chennai)

Programme Committee

Deeparnab Chakrabarty (MSR, India)
Timothy Chan (Univ. Waterloo)
Anirban Dasgupta (IIT Gandhinagar)
Amit Kumar (IIT Delhi)
Neeldhara Misra (IISc, Bangalore)
Subhas Nandy (ISI Kolkata)
Patrick Nicholson (MPI Saarbrücken)
Michał Pilipczuk (Univ. Bergen)
Jaikumar Radhakrishnan (TIFR, Mumbai)
Rajmohan Rajaraman (Northeastern Univ.)
Rahul Santhanam (Univ. Edinburgh)
Jayalal Sarma (IIT Madras)
Srikanth Srinivasan (IIT Bombay)
Kavitha Telikepalli (TIFR, Mumbai)
Suresh Venkatasubramanian (Univ. Utah)

Dietmar Berwanger (LSV, ENS Cachan)
Ahmed Bouajjani (LIAFA Paris)
Supratik Chakraborty (IIT Bombay)
Radha Jagadeesan (DePaul University)
Aditya Kanade (IISc, Bangalore)
Dietrich Kuske (TU Ilmenau)
Ralf Küsters (Univ. Trier)
Kamal Lodaya (IMSc, Chennai)
Madhavan Mukund (CMI, Chennai)
Shaz Qadeer (MSR, Redmond)
Arnaud Sangnier (LIAFA Paris)
Alexis Saurin (CNRS, Univ. Paris-Diderot)
Sunil Simon (IIT Kanpur)
Ashutosh Trivedi (IIT Bombay)
Mahesh Viswanathan (Univ. Illinois)

Organizing Committee

Shweta Agrawal (IIT Delhi)
Naveen Garg (IIT Delhi)
Ragesh Jaiswal (IIT Delhi), chair
Amit Kumar (IIT Delhi)
Sanjiva Prasad (IIT Delhi)
Sandeep Sen (IIT Delhi)

External Reviewers

Accattoli, Beniamino	Akshay, S.
Antoniadis, Antonios	Apt, Krzysztof
Arun-Kumar, S.	Asarin, Eugene
Atig, Mohamed Faouzi	Baelde, David
Basavaraju, Manu	Baskar, A.
Basset, Nicolas	Batra, Jatin
Bauer, Matthew S.	Bhattacharya, Bhaswar
Bhattacharya, Sayan	Biondi, Fabrizio
Bollig, Benedikt	Brenguier, Romain
Brunet, Paul	Böhm, Stanislav
Chadha, Rohit	Chandran, Nishanth
Chatterjee, Krishnendu	Chen, Taolue
Chroboczek, Juliusz	Curticapean, Radu
D'Souza, Deepak	Darbari, Ashish
Das, Bireswar	Das, Gautam
Dawar, Anuj	de Keijzer, Bart
De Rougemont, Michel	de Souza, Rodrigo
De, Minati	Decker, Normann
Dinesh, Krishnammorthy	Dragoi, Cezara
Drange, Paal Groenaas	Duggirala, Parasara Sridhar
Durand-Gasselin, Antoine	Dzhafarov, Damir
Díaz-Caro, Alejandro	Eisentraut, Christian
Elberfeld, Michael	Enduri, Murali
Fearnley, John	Forejt, Vojtech
Fournier, Hervé	Gagie, Travis
Garg, Ankit	Gawrychowski, Pawel
Geeraerts, Gilles	Genest, Blaise
Goswami, Mayank	Grädel, Erich
Gupta, Manoj	Gupta, Rahul
Gutierrez, Julian	Gutin, Gregory
Habermehl, Peter	Hague, Matthew
Hansen, Thomas Dueholm	Harsha, Prahladh
Herbreteau, Frédéric	Hlineny, Petr
Holzer, Markus	Huschenbett, Martin
Jain, Rahul	Kaleeswaran, Shalini
Karandikar, Prateek	Karmarkar, Hrishikesh
Kartzow, Alexander	Kasiviswanathan, Shiva
Kini, Dileep	Klaedtke, Felix
Klimann, Ines	Knapik, Teodor
Knight, Sophia	Kobayashi, Yusuke
Kociumaka, Tomasz	Koepf, Boris
Komarath, Balagopal	Koroth, Sajin
Kratsch, Stefan	Krithivasan, Kamala
Krivine, Jean	Kulkarni, Janardhan
Kumar, Ravi	Kupferman, Orna
Kurur, Piyush	Künnemann, Marvin

Löding, Christof
Lee, Min-Joong
Limaye, Nutan
Lombardy, Sylvain
Madnani, Khushraj
Manthey, Bodo
Mardare, Radu
Mazowiecki, Filip
Meyer, Antoine
Mikučionis, Marius
Monmege, Benjamin
Morgan, Carroll
Mueller, Moritz
Mukhopadhyay, Sagnik
Müller-Olm, Markus
Narayanaswamy, N. S.
Nickovic, Dejan
Nori, Aditya
Novotný, Petr
Papanikolaou, Nick
Parlato, Gennaro
Parys, Pawel
Peressotti, Marco
Philip, Geevarghese
Podelski, Andreas
Prasad, Sanjiva
Pédrot, Pierre-Marie
Ramanujam, R.
Rao B. V., Raghavendra
Reidl, Felix
Rosa-Velardo, Fernando
Roy, Sambuddha
S., Krishna
Sankur, Ocan
Santocanale, Luigi
Sarpatwar, Kanthi
Saurabh, Nitin
Scapin, Enrico
Schmitz, Sylvain
Sen, Pranab
Seth, Anil
Sharma, Roohani
Shpilka, Amir
Srivathsan, B.
Straubing, Howard
Sunil, K. S.
Thinniyam, Ramanathan
Trivedi, Ashutosh
Le Roux, Stephane
Li, Shi
Lohrey, Markus
M. S., Ramanujan
Maiya, Pallavi
Manuel, Amaldev
Mathur, Umang
Mennicke, Roy
Mihalák, Matúš
Milanic, Martin
Montes, Pablo
Mount, David
Mukherjee, Suvam
Murawski, Andrzej
Narayan Kumar, K.
Nasre, Meghana
Nimbhorkar, Prajakta
Norman, Gethin
Pandya, Paritosh
Park, Sungwoo
Parthasarathy, Madhusudan
Paul, Soumya
Pettie, Seth
Pilipczuk, Marcin
Prabhakaran, Vinod M.
Praveen, M.
Raghothaman, Mukund
Ramesh, S.
Reddy, Vinod
Roohi, Nima
Rossman, Benjamin
Rubin, Sasha
Salinger, Alejandro
Santhiar, Anirudh
Saptharishi, Ramprasad
Satti, Srinivasa Rao
Saurabh, Saket
Schlotter, Ildikó
Schnoor, Henning
Sen, Sagnik
Shah, Simoni
Sherkhonov, Evgeny
Sreejith, A. V.
Steinberger, John
Sundararajan, Vaishnavi
Swamy, Nikhil
Torán, Jacobo
Truderung, Tomasz

Tulsiani, Madhur	Tzameret, Iddo
Tzevelekos, Nikos	Vaish, Rohit
Valiron, Benoît	Varacca, Daniele
Venkat, Rakesh	Volkov, Mikhail
Walukiewicz, Igor	Watrous, John
Wilke, Thomas	Wojtczak, Dominik
Wolf, Karsten	Worrell, James
Wu, Bang Ye	Yehudayoff, Amir
Zehavi, Meirav	Zetzsche, Georg
Zeume, Thomas	Živný, Stanislav

New Developments in Iterated Rounding*

Nikhil Bansal

Eindhoven University of Technology
Eindhoven, The Netherlands
`n.bansal@tue.nl`

Abstract

Iterated rounding is a relatively recent technique in algorithm design, that despite its simplicity has led to several remarkable new results and also simpler proofs of many previous results. We will briefly survey some applications of the method, including some recent developments and giving a high level overview of the ideas.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Algorithms, Approximation, Rounding

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.1

Category Invited Talk

1 Introduction

A natural and very general approach to designing approximation algorithms for NP-Hard problems is the following: Write an exact integer programming formulation for the problem, and then relax the integer constraints to be continuous, giving rise to a convex optimization problem such as a linear program or a semidefinite program that can be solved efficient in polynomial time. The goal then is to design a good *rounding* procedure that converts this fractional solution back to an integral solution without much loss in the value of the objective.

In the last few decades various ingenious rounding techniques have been developed, with several surprising connections to probability, geometry and so on. For an overview of these methods and approximation algorithms in general, we refer the reader to [10]. In recent years, a powerful new approach for rounding, referred to as iterated rounding has emerged. Here the variables are rounded one by one, or in small steps over time, while crucially leveraging the information gained from previous steps. While this idea is not new by itself, many interesting and surprising applications have emerged recently. An excellent description of iterated rounding and its applications can be found in [5].

Here we give a brief introduction to the method. We start with some classic results to demonstrate its versatility and power, and then discuss some more recent developments and incarnations based on techniques such as discrepancy and Lovász Local Lemma.

2 The Basic Approach

Consider a linear programming relaxation of the form

$$\min c^T x \qquad \text{s.t.} \quad Ax \le c \qquad \text{and} \quad x \in [0,1]^n,$$

* Supported by NWO grant 639.022.211 and ERC consolidator grant 617951.

with variables $x_1, \ldots, x_n$. Here x denotes the vector $(x_1, \ldots, x_n)$ and A is some $m \times n$ matrix, and c is a non-negative cost vector in R^n.

We will refer to the m constraints given by rows of $Ax \leq c$ as the non-trivial constraints, and the remaining constraints given by $x \in [0,1]^n$ as the trivial constraints.

Recall that for any linear program, there is always some optimum solution that lies at the vertex of the polytope formed by the constraints. Such a solution is referred to as a basic feasible solution. The key idea behind much of iterated rounding is the following easy observation.

▶ **Lemma 1.** *For any linear program of the form above with $m < n$ non-trivial constraints, there is an optimum solution with at least $n - m$ variables set to 0 or 1.*

Proof. Given a solution x, We say that a constraint is tight at x if it is satisfied by equality by at x. As the polytope in n-dimensional, every vertex x of the polytope is determined uniquely by some n linear independent constraints that are tight at x.

Consider some basic feasible optimal solution. As there are m non-trivial constraints, at least $n - m$ of the tight constraints at x must be trivial ones. If a trivial constraint involving x_i is tight, this means that $x_i = 0$ or $x_i = 1$. ◀

The algorithm proceeds as follows:

1. Start with the initial LP, and compute some basic feasible solution x^*.
2. Permanently fix the value of any variable that is set to 0 or 1, and then consider the residual LP (obtained by drop this fixed variable and updating the right hand of each constraint accordingly).
3. Find a linearly independent set of tight constraints at x^*, and choose one (or more) of these constraints in some suitable problem specific manner (this is where the ingenuity lies) and drop it from the residual LP. Recompute a basic feasible solution of this reduced LP and iterate the process until an integer solution is obtained.

The key observation is that dropping one of the constraints that determines x^* allows the LP to get *unstuck* at x^* and move to some another vertex solution. Note that since we drop at least one constraint at each iteration, the procedure will eventually terminate. Moreover, it is easy to verify that objective value of the LP can only go down during the various iterations of the algorithm (as permanently fixing a variable that is already 0 or 1 does not affect the objective value, and dropping a constraint can only reduce the objective).

We begin with a simple example.

2.1 Makespan Minimization on Unrelated Machines

The unrelated machine setting is the following. There are m machines and n jobs. Each job j must be processed on some machine, and it has arbitrary machine dependent processing time p_{ij} on machine i. The goal is to assign jobs to the machines to minimize the makespan (or the maximum load over all machines). In a classic result, Lenstra, Shmoys and Tardos [6] gave a 2-approximation for the problem, and also show that no $1.5 - \epsilon$ approximation exists unless P=NP. It is a major open question in approximation algorithms to improve the approximation ratio of 2. Here we give a simple iterated rounding based proof of their result, as described in [5].

2.1.1 Algorithm

By doing a binary search we can assume that we know the value T of the optimum makespan. We will write a LP with variables x_{ij} with the intended solution that $x_{ij} = 1$ if j is assigned to machine i and 0 otherwise.

We do an initial preprocessing step where we set $x_{ij} = 0$ if $p_{ij} > T$ (as j can never be assigned to i in a solution with makespan T). Let us assume that we are given a feasible solution to the following LP.

$$\sum_j p_{ij}x_{ij} \leq T \qquad \forall i \in [m] \tag{1}$$

$$\sum_i x_{ij} = 1 \qquad \forall j \in [n] \tag{2}$$

$$x_{ij} \geq 0 \qquad \forall i, j \tag{3}$$

Note that while the number of variables can be nm, the number of non-trivial constraints (1) and (2) is $n + m$. Also note that we do not impose the constraint $x_{ij} \leq 1$ as this is implied by (2).

The algorithm proceeds via the iterated rounding framework. Recall that all we need to do is to design a procedure that given a basic feasible solution as input, determines which constraint to drop in the current round.

Consider some basic feasible solution to the LP in the current round. We fix the variables that are already 0 or 1, and consider the residual solution on variables x_{ij} with $0 < x_{ij} < 1$. We say that j appears on i if $x_{ij} > 0$. We will show the following.

▶ **Lemma 2.** *There exists a machine i such that (i) exactly one job j appears on i, or (ii) exactly two jobs j and j' appear on i and satisfy $x_{ij} + x_{ij'} \geq 1$.*

Given Lemma 2, the dropping rule will be to drop the load constraint (1) corresponding to this machine i. Lemma 2 ensures that no matter how the variables are rounded in subsequent iterations the additional load assigned to machine i can be at most $p_{\max}$. Indeed, in case (i) either job j can be assigned to i, which can increase the load by at most $(1 - x_{ij})p_{ij} \leq p_{ij}$, and in case (ii) both j and j' can be assigned to i, and the load can increase by at most $p_{ij}(1 - x_{ij}) + p_{ij'}(1 - x_{ij'}) \leq p_{\max}(2 - x_{ij} - x_{ij'}) \leq p_{\max}$.

It remains to show Lemma 2. This is done by a counting argument (which is typical in most iterated rounding proofs).

Proof. Let p (resp. f) denote the number of variables x_{ij} with $0 < x_{ij}$ (resp. $0 < x_{ij} < 1$). As the number of non-trivial constraints is at most $n + m$ (and as the trivial constraints are of the form $x_{ij} \geq 0$), there is a basic feasible solution with $p \leq n + m$. Note that each job contributes 1 to p if it is assigned integrally to some machine, and least 2 if it is assigned fractionally to two or more machines. So we obtain that

$$n \leq (p - f) + (f/2) = p - f/2 \leq (n + m) - f/2. \tag{4}$$

The first inequality follows as $p - f$ is the number of variables with $x_{ij} = 1$ and the last inequality follows as $p \leq n + m$. This gives that $f \leq 2m$. If $f \leq 2m - 1$, then we are already done as there must exist some machine i with at most one fractional variable appearing on it.

On the other hand, if exactly two fractional variables appear on each machine, then $f = 2m$, which implies that equality must hold throughout in (4), and hence $p - f = n - m$. This implies that exactly m jobs are split fractionally, and each of them appears on exactly two machines. Thus, there must be some machine i with two jobs j and j' with $x_{ij} + x_{ij'} \geq 1$, as claimed. ◀

2.2 Degree-bounded Spanning Trees

Our next example is one where on first glance it would seem that iterated rounding should not work, as the number of non-trivial constraints m is substantially larger than the number of variables n, and the conditions of Lemma 1 do not seem to apply.

The minimum cost degree bounded spanning tree problem is defined as follows. Given a graph $G = (V, E)$ with non-negative edge costs c_e, and degree bounds b_v on the vertices, find a minimum cost spanning tree of G satisfying the degree bounds. Even though the minimum spanning tree problem is efficiently solvable, adding the degree bounds makes the problem NP-complete. In particular, if $b_v = 2$ for all v, the problem reduces to Hamiltonian Path.

We will show the following result of Singh and Lau [9] which gives essentially the best possible guarantee.

▶ **Theorem 3.** *There is an efficient algorithm that finds a spanning tree with degree bounds violated by at most $+1$, and with cost at most the cost of optimum spanning tree (that satisfies the degree bounds exactly).*

2.2.1 Algorithm

The starting point is the following natural LP relaxation for the problem with variables x_e that are supposed to indicate whether edge e is chosen or not.

$$\begin{aligned}
& \min c_e x_e && && (5)\\
\text{s.t.}\quad & \sum_{e\in E[S]} x_e && \leq |S|-1 \qquad \forall S \subset V && (6)\\
& \sum_{e} x_e && = n-1 && (7)\\
& \sum_{e\in\delta(v)} x_e && \leq b_v \qquad \forall v \in v && (8)\\
& 0 \leq x_e && \leq 1 \qquad \forall e \in E && (9)
\end{aligned}$$

Here $E[S]$ is the set of edges in the subgraph induced by $S \subset V$, and $\delta(v)$ is the set of edges incident to v. Recall that the constraints (6), (7) and (9) completely characterize the spanning tree polytope.

Observe that the number of variables is $O(n^2)$ (one for each edge) while the number of constraints (6) are exponentially many. Still it turns out that the iterated framework can be applied very effectively.

The crucial observation is that even though exponentially many constraints (6) may be tight at a given vertex of the polytope, all these tight constraints are spanned by a small set of at most $n-1$ linearly independently constraints. More precisely, the supermodularity of the function $\chi_{E[S]}$ implies that the tight constraints given by (6) can be uncrossed, and hence spanned by constraints corresponding to sets forming a laminar family. Thus the fractional part of the basic feasible solution is completely determined by these (at most) $n-1$ constraints together with some other tight degree constraints. As the number of degree constraints can be at most n, essentially the number of relevant constraints (and hence the number of fractional variables) is at most $(n-1)+n$. We remark later in later iterations, a slightly more careful argument is needed.

A counting argument now allows one to show that there is some vertex v such that $\sum_{e\in\delta(v):x_e>0}(1-x_e) < 2$. The details are quite simple, and we refer the reader to [5] for details.

This implies that if we drop the degree constraint on vertex v, then even if all the variables are rounded to 1 is subsequent iterations, the degree violation can be at most strictly less than 2. This specifies the dropping rule for the algorithm. The algorithm keeps iterating until it has found an integral solution, or until all degree bounds have been dropped in which case the LP reduces to the spanning tree LP which is integral. Finally, we observe that since the degrees and integral, a violation of strictly less than 2 implies a violation of at most 1.

2.3 Bin Packing

Our next example gives an application of iterated rounding where we drop a constant fraction of constraints at each iteration, and the algorithm terminates in a logarithmic number of rounds.

The classical bin packing problem is the following. Given a collection of items with sizes $s_1, \dots, s_n$ where each $0 < s_i \leq 1$, pack these items feasibly into the fewest number of unit size bins. The problem is NP-Complete, as the Partition problem implies that it is hard to distinguish if the optimum packing requires 2 bins or 3 bins. In fact, this is the best known hardness for the problem, and it is a major open question to determine whether there exists a polynomial time algorithm achieving an $\mathrm{Opt} + O(1)$ or even $\mathrm{Opt} + 1$ guarantee.

In 1981, Karmarkar and Karp [4] gave a remarkable algorithm that achieves a guarantee of $\mathrm{Opt} + O(\log^2 \mathrm{Opt})$. This result is one of the first applications of iterated rounding.

2.3.1 Algorithm

The starting point is a very strong relaxation known as the configuration LP. Suppose we have an instance that contains n_i items of size s_i. Let k denote the number of distinct sizes. By simple arguments, we can ignore items of size $\leq 1/\mathrm{Opt}$ (as these items are easy to fill in later). A valid configuration C is any multiset of sizes in the collection $\{s_1, \dots, s_k\}$ with total size at most 1. Let $\mathcal{C}$ denote the collection, possibly exponentially large, of all valid configurations.

Consider the following LP formulation with a variable x_C for each configuration $C \in \mathcal{C}$, that is supposed to indicate the number of bins packed using configuration C.

$$\min \sum_C x_C \qquad \text{s.t.} \quad \sum_C a_{i,C} x_C \geq n_i \quad \forall i = 1, \dots, k, \quad \text{and} \quad x_C \geq 0 \qquad \forall C \in \mathcal{C}.$$

Here $a_{i,C}$ denotes the number of items of size s_i in C. Even though the number of variables is exponential, this LP can be solved to any desired accuracy by considering the dual (and moreover a basic feasible solution can be computed using standard techniques). As this LP has only $k \leq n$ constraints, at most k configurations C have non-zero x_C values.

The crucial observation of [4] was the following.

▶ **Lemma 4.** *Given a bin packing instance I with total size of items $s(I)$, there is a procedure to round up the size of each item sizes to obtain another instance $\tilde{I}$ with at most $s(I)/2$ distinct item sizes, and $Opt(\tilde{I}) \leq Opt(I) + O(\log s(I))$.*

The proof of Lemma 4 is quite simple and we refer the reader to [5] for a proof.

Let us see how this gives the claimed algorithm. Consider an instance with k item sizes. The algorithm solves the configuration LP to obtain a solution with at most k non-zero configurations. For each C with $x_C > 0$, pack $\lfloor x_C \rfloor$ bins with configuration C and remove these items. Consider the instance consisting of the remaining unpacked items. As these fit fractionally in at most k configurations, the total size of these items is at most k, and by

Lemma 4 we can round these to $k/2$ sizes, while losing at most $O(\log k)$ in the objective. We now iterate the algorithm on this rounded instance.

Observe that at each iteration the number of distinct sizes decreases by at least half, and thus there are logarithmically many iterations, each adding at most $\log(\mathrm{Opt})$ to the objective. This implies the claimed result.

2.4 Flow Time Minimization

Usually when designing a LP based approximation, one tries to add as many valid constraints as possible to make the relaxation tighter. However, sometimes it is beneficial to reduce the number of constraints so that iterated rounding can be used. We next give an example of a problem for which strong LP relaxations are known but they key is to consider a different (weaker) formulation with much fewer constraints.

The problem is that of minimizing the total flow time on unrelated machines. Given a collection on n jobs and m machines, where job j has size p_{ij} on machine i and release time r_j, find a feasible schedule to minimize the total flow time. Here the flow time of a job is the amount of time it spends in the system; i. e., its completion time minus its arrival time. We assume that the schedule is non-migratory and preemptive, i. e., a job can only be executed on a single machine and can be preempted and resumed later without any penalty.

Recently, Bansal and Kulkarni [2] gave the first poly-logarithmic approximation for the problem based on iterated rounding.

2.4.1 Standard LP formulation

We first describe the widely used standard time indexed LP relaxation for our problem. There is a variable x_{ijt} for each machine $i \in [m]$, each job $j \in [n]$ and each unit time slot $t \geq r_j$. The x_{ijt} variables indicate the amount to which a job j is processed on machine i during the time slot t. The first set of constraints (10) says that every job must be completely processed. The second set of constraints (11) says that a machine cannot process more than one unit of job during any time slot. The objective function is referred to as the fractional flow time and will be irrelevant to our discussion here.

$$\min \sum_{i,j,t} \left(\frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot x_{ijt}$$

$$\text{s. t.} \quad \sum_i \sum_{t \geq r_j} \frac{x_{ijt}}{p_{ij}} \geq 1 \qquad \forall j \tag{10}$$

$$\sum_{j \,:\, t \geq r_j} x_{ijt} \leq 1 \qquad \forall i, t \tag{11}$$

$$x_{ijt} \geq 0 \qquad \forall i, j, t \geq 0$$

2.4.2 New LP formulation

We now describe a new LP relaxation for the problem, where we do not enforce the capacity constraints (11) for each time slot, but instead only enforce these constraints over carefully chosen intervals of time.

Let $P = \max_{i,j} p_{ij} / \min_{i,j} p_{ij}$ and assume that $\min_{i,j} p_{ij} = 1$. For $k = 0, 1, \ldots, \log P$, we say that a job j belongs to class k on machine i if $p_{ij} \in (2^{k-1}, 2^k]$. Note that the class of a job depends on the machine.

There is a variable y_{ijt} (similar to x_{ijt} before) that denotes the total units of job j processed on machine i at time t. However, unlike the time indexed relaxation, y_{ijt} is allowed to take values greater than one.

For each class k and each machine i, we partition the time horizon $[0, T]$ into intervals of size $4 \cdot 2^k$. For $a = 1, 2, \ldots$, let $I(i, a, k) = ((4 \cdot 2^k)(a-1), (4 \cdot 2^k)a]$ denote the a-th interval of class k on machine i. The new relaxation is the following.

$$\sum_i \sum_{t \geq r_j} \sum_k \sum_{j \in (2^{k-1}, 2^k]} \left(\frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot y_{ijt}$$

$$\begin{aligned} \text{s. t.} \quad \sum_i \sum_{t \geq r_j} \frac{y_{ijt}}{p_{ij}} &\geq 1 \qquad \forall j && (12) \\ \sum_{j \,:\, p_{ij} \leq 2^k} \sum_{t \in I(i,a,k)} y_{ijt} &\leq \text{Size}(I(i,a,k)) \qquad \forall i, k, a && (13) \\ y_{ijt} &\geq 0 \qquad \forall i, j, t \,:\, t \geq r_j \end{aligned}$$

Here, Size$(I(i, a, k))$ denotes the size of the interval $I(i, a, k)$ which is $4 \cdot 2^k$ (but would change in later iterations of the LP when we apply iterated rounding). Observe that in (13) only jobs of class $\leq k$ contribute to the left hand side of constraints corresponding to intervals of class k.

The main idea why this LP is useful is the following. For simplicity assume that all jobs belong to a single class k. Some some basic feasible solution and suppose that h constraints given by (13) are tight. As there are n constraints 12, at most $n + h$ non-trivial constraints can be tight and hence at most $n + h$ variables are non-zero. If h constraints (13) are tight, this also means that the total size of jobs is at least $h \cdot (4 \cdot 2^k)$. As these are class k-jobs, this means that $n \geq 4h$, and hence the number of non-zero variables is at most $n + h \leq 5/4n$. So a constant fraction of the jobs j must be assigned integrally to a single machine. Thus after logarithmic iterations, this produces a reasonable *pseudo-schedule*, that can be converted to a proper schedule without much additional loss. We refer the reader to [2] for details.

3 A Generalization based on Discrepancy

Recently a very powerful generalization of iterated rounding was developed, based on discrepancy theory. The examples of iterated rounding that we have seen thus far are based on Lemma 1 which requires that $m < n$, and to maintain this invariant, in each the algorithm drops certain constraints in each iteration. Observe that when a constraint $a_j x \leq b_j$ is dropped, we have no control on how much this constraint could be violated in future iterations. In particular the violation could be as large as $\|a_j\|_1$ (e. g. if all the variables x_i are very close to 0 when the constraint was dropped, and eventually they all get rounded to 1).

Recently, Lovett and Meka [7] gave the following rounding result.

▶ **Theorem 5.** *Let $x \in [0, 1]^n$ be some fractional solution to a linear system $Ax = b$, where A is an $m \times n$ matrix. For $j = 1, \ldots, m$, let λ_j be such that*

$$\sum_j \exp(-\lambda_j^2/16) \leq n/16. \tag{14}$$

Then there is an efficient algorithm to find a solution $\tilde{x}$ with the following properties:
(i) *at most $n/2$ variables fractional (that is strictly between 0 and 1),*
(ii) *$|a_j \tilde{x} - a_j x| \leq \lambda_j \sqrt{\|a_j\|_2}$ for each $j = 1, \ldots, m$, where a_j denotes the j-th row of A.*

Let us parse what theorem 5 gives us. First observe that if $m \leq n/16$, then setting each $\lambda_j = 0$ for $j = 1, \ldots, m$ satisfies (14), and gives a solution $\tilde{x}$ that does not violate any constraint and has $n/2$ variables integral.

In this setting Lemma 1 would give a solution with $n - m = (15/16)n$ variables set integrally and no constraint violated. Thus ignoring constants (which can be modified if needed by the application at hand, see for example [1]) this can be viewed as an analog of Lemma 1.

However, theorem 5 also holds when $m \gg n$ provided the error parameters λ_j are chosen to satisfy condition (14). The fact that one has complete flexibility in how to choose λ_j (provided (14) holds) can make this variant extremely powerful. For example suppose $m = 10n$. Then, standard iterated does not give anything until $m - n = 9n$ more constraints are dropped, potentially incurring $\|a_j\|_1$ error on these dropped constraints . On the other hand one can set each λ_j to be $O(1)$ in theorem 5 to obtain error $O(\|a_j\|_2)$ for each row j. The crucial point is that the ℓ_2 norm $\|a_j\|_2$ of a constraint can be substantially smaller than its ℓ_1 norm $\|a_j\|_1$ (e. g. $\sqrt{n}$ vs n), and hence theorem 5 can give much less error. Moreover, by setting $\lambda_j's$ non-uniformly one can enforce smaller error on more critical constraints.

3.1 Improvement for Bin Packing

Recently, based on these ideas, Rothvoss [8] improved the long-standing bound of $\mathrm{Opt} + O(\log^2(\mathrm{Opt}))$ for bin packing that we saw in Section 2.3 to $\mathrm{Opt} + O(\log \mathrm{Opt} \log\log \mathrm{Opt})$. The main observation was that if most of the configurations are “well-spread", that is, they do not consist of only few item types that appear many times, then the ℓ_2 norm of the vector corresponding to such a configuration (i. e. the vector indicating how many times each item type appears) is much smaller than its ℓ_1 norm. However, there is no apriori reason why such configurations should be used by a basic feasible solution to the LP. To get around this problem, Rothvoss introduces the crucial idea of a creating new items by grouping together various small items of the same size that appear in a configuration. These ideas together with the framework of Karmarkar Karp then give the improved bound. The details are somewhat technical and we refer the reader to [8] for details.

4 Iterated rounding via the Lovász Local Lemma

Our final example illustrates how other powerful tools such as the Lovász Local Lemma can fit nicely into the iterative approach for rounding.

The problem we consider is that of minimizing makespan on unrelated machines in the multi-dimensional setting. There are m machines and n jobs. Each machine has d resources and each job needs some quantities of these resources. For example, the machines could correspond to computers and the $d = 2$ resources could be CPU and memory. In the unrelated machines setting, the load of job j on machine i is specified by an arbitrary non-negative number p_{ijk} for each $k \in [d]$. In typical scenarios d is a fixed small constant, and n and m are much larger.

As in Section 2.1, we can guess the optimum makespan T, and write an LP with variables x_{ij}. We set $x_{ij} = 0$ if $p_{ijk} > T$ for some k, and find a feasible solution to the following.

$$\sum_j p_{ijk} x_{ij} \leq T \qquad \forall i \in [m], k \in [d] \tag{15}$$

$$\sum_i x_{ij} \geq 1 \qquad \forall j \in [n] \tag{16}$$

Let us first observe what the natural approaches give. As the LP has $n + md$ constraints, each machine could potentially have $\Omega(d)$ jobs fractionally assigned to it in a basic feasible solution. So applying the approach in Section 2.1 only gives an $O(d)$ approximation.

On the other hand if we do randomized rounding (i. e. independently assign each job j to machine i with probability x_{ij}), then by a standard balls in bins argument, the makespan could be as high as $\Theta(T \log dm/ \log\log dm)$, which has an undesirable dependence on m.

It turns out that one can show the following substantially stronger result.

▶ **Theorem 6** ([3]). *There is an $O(\log d/ \log\log d)$ approximation for the problem.*

Before we sketch the idea behind theorem 6, we recall the Lovász Local Lemma.

▶ **Lemma 7.** *Let $B_1, B_2, \ldots, B_k$ be a collection of (bad) events such that each B_i occurs with probability at most p and is independent of all the other events except for at most d of them. If epd < 1, then there is a nonzero probability that none of the events occurs.*

The idea of the algorithm is to start with an arbitrary solution x, and gradually make the variables x_{ij} *closer* to integral in each iteration, without substantially deteriorating the quality of the solution. In particular, in iteration ℓ, the values x_{ij} will be integral multiples of ϵ_ℓ, where ϵ_ℓ increases exponentially with ℓ, until $\epsilon_\ell = \Omega(\log\log d/ \log d)$. At this point that algorithm can arbitrarily assign a job j to any machine i where $x_{ij} > 0$.

Let $\epsilon_0 = 1/(\log dm)$. Given an initial solution x, we can assume that each x_{ij} is an integral multiple of ϵ_0 by rounding each x_{ij} independently to either $\epsilon_0\lfloor x_{ij}/\epsilon_0\rfloor$ or $\epsilon_0\lceil x_{ij}/\epsilon_0\rceil$, with the right probability such that its expectation remains the same. By standard Chernoff bounds the makespan remains $O(T)$ with high probability, and we can further scale the solution by an $O(1)$ factor to ensure that $\sum_i x_{ij} \geq 1$ for each job j. Let us also assume here (to avoid some technical details), that each job j is *large* on every machine i in the sense that the ℓ_1 norm of its load vector satisfies $\sum_k p_{ijk} \geq 1/d$.

Consider round ℓ, where we round the values of $x_{ij}^{\ell-1}$ (as previously) at the end of iteration $\ell - 1$, to multiples of some suitably chosen $\epsilon_\ell \gg \epsilon_{\ell-1}$. The following observation is crucial.

▶ **Lemma 8.** *Consider round ℓ and apply the rounding to $x_{ij}^{\ell-1}$ mentioned above to obtain x_{ij}^{ℓ}. Let B_i be the event that the load on machine i increases by more than T for some coordinate k, and let A_j denote the event that for a job $\sum_i x_{ij}^{\ell} \leq 1/2$. Then each of these events depends on at most poly$(d, \epsilon_{\ell-1})$ other events.*

Proof. Two events B_i and $B_{i'}$ are dependent if and only if $x_{ij}^{\ell-1} > 0$ and $x_{i'j}^{\ell-1} > 0$ for some job j. Similarly, two events A_j and B_i are dependent if and only if some $x_{ij}^{\ell-1} > 0$. As each $x_{ij}^{\ell-1}$ is an integral multiple of $\epsilon_{\ell-1}$ and each job is large, the total number of jobs assigned to a machine can be at most $O(d^2 \cdot (1/\epsilon_{\ell-1}))$. Moreover, for a job j at most $O(1/\epsilon_\ell)$ variables $x_{ij}^{\ell-1}$ can be non-zero. Together this implies the claim. ◀

Thus by Lemma 7, we can choose $\epsilon_\ell = \Theta(\log\log(d/\epsilon_{\ell-1})/ \log(d/\epsilon_\ell))$ and ensure that none of the bad event happens. Thus crucial observation is that as we iterate the algorithm, the dependence on m in ϵ_ℓ becomes like $\log^{(\ell)} m$ (i. e. log iterated ℓ times) and eventually disappears[1], while the dependence on d converges to $\Omega(\log\log d/ \log d)$.

[1] Strictly speaking, this gives an $\exp(O(\log^* m))$ guarantee. But a more gradual rounding with a slightly more careful of parameters gives theorem 6. We refer to [3] for details.

References

1 Nikhil Bansal, Moses Charikar, Ravishankar Krishnaswamy, and Shi Li. Better algorithms and hardness for broadcast scheduling via a discrepancy approach. In *SODA*, pages 55–71, 2014.

2 Nikhil Bansal and Janardhan Kulkarni. Minimizing flow-time on unrelated machines. *CoRR*, abs/1401.7284, 2014.

3 David G. Harris and Aravind Srinivasan. The moser-tardos framework with partial resampling. In *FOCS*, pages 469–478, 2013.

4 Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *FOCS*, pages 312–320, 1982.

5 Lap-Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, 2011.

6 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.

7 Shachar Lovett and Raghu Meka. Constructive discrepancy minimization by walking on the edges. In *FOCS*, pages 61–67, 2012.

8 Thomas Rothvoss. Approximating bin packing within o(log OPT * log log OPT) bins. In *FOCS*, pages 20–29, 2013.

9 Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *STOC*, pages 661–670, 2007.

10 David Williamson and David Shmoys. *The design of Approximation Algorithms*. Cambridge University Press, 2011.

Reasoning About Distributed Systems: WYSIWYG*

Aiswarya Cyriac[1] and Paul Gastin[2]

1 Uppsala University, Sweden
aiswarya.cyriac@it.uu.se

2 LSV, ENS Cachan, CNRS, INRIA, France
paul.gastin@lsv.ens-cachan.fr

Abstract

There are two schools of thought on reasoning about distributed systems: one following interleaving-based semantics, and one following partial-order/graph-based semantics. This paper compares these two approaches and argues in favour of the latter. An introductory treatment of the split-width technique is also provided.

1998 ACM Subject Classification F.1.1 [Computation by Abstract Devices]: Models of Computation, F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs

Keywords and phrases Verification of distributed systems, Communicating recursive programs, Partial order/graph semantics, Split-width and tree interpretation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.11

Category Invited Talk

1 Introduction

Distributed systems form a crucially important but particularly challenging domain. Designing correct distributed systems is demanding, and verifying its correctness is even more so. The main cause of difficulty here is concurrency and interaction (or communication) between various distributed components. Hence it is important to provide a framework that makes easy the design of systems as well as their analysis. In this paper we argue in favour of (visual) graph-based techniques towards this end.

The behaviour of a distributed system is often understood by means of an interleaving-based semantics. People are guided by this understanding when designing a system, and also when formally expressing properties for system verification. But interleavings obfuscate the interactions between components. This inherent complication of interleaving-based semantics makes the design and verification vulnerable to many (human) errors. Moreover, expressing distributed properties on interleavings is non-trivial and sometimes also impossible to achieve. In contrast, a visual understanding of behaviours of distributed systems would make it less prone to errors – in the understanding of the semantics, in the statement of properties and in verification algorithms. Not only does the visual approach help in providing right intuitions, but, we will demonstrate that, it is also very powerful and efficient.

* Supported by LIA InForMel.

What you see is what you understand. A good example of the visual representation of behaviours is the ITU standard message sequence charts (MSCs) [20] which describe protocols involving message exchanges. The events executed by a local process are linearly ordered, see e. g., Figure 3. The transmission of the messages is also depicted. This visual description reveals not only the interactions between components but also the concurrency and the causality relations. The causality relation corresponds to the transitive closure of the linear orders on processes and of the message relation. Events that are not causally ordered are concurrent.

Another example which illustrates the power of visual representations is nested words [5] for the behaviours of recursive programs. The binary relation matching push-pop pairs, which is very fundamental for reasoning about recursive (or pushdown) systems, is explicitly provided in a nested word.

What you see is what you state. One main advantage of such a visual representation is the ease and power of specification. The underlying graph structure provides a richer framework for formal specification. For example, monadic second order logic (MSO) may have causality relation, concurrency relation, process ordering, message transmissions, push-pop matching relation etc. as basic predicates. Many of these fundamental relations are very difficult (or even impossible) to recover if we settle for an interleaving-based understanding. For example, the monadic second order logic over words cannot express a matching push-pop relation even when we assume a visible alphabet (one in which letter dictates whether it is a push position or a pop position). This is because such a relation requires some implicit counting for which MSO over words is too weak.

What about LTL? Temporal logics and navigation logics have been studied over the intuitive visual descriptions, for instance over nested words [4, 3], MSCs [8], nested traces [7], multiply nested words [24], etc. Such logics allow us to express the fundamental relations (causality, concurrency, message matching, push-pop matching, etc.) as basic modalities. Thus, properties of distributed systems can be easily and naturally expressed. On the other hand, if the behaviour of a distributed system is understood in terms of linear sequences of events, one is tempted to use LTL over words for specifications. But the classical modalities of LTL are not suited to the distributed setting. For example, the temporal next modality of LTL over words is nonsensical in the actual distributed behaviour since concurrent/independent events can be ordered arbitrarily by the operational semantics. So, LTL is sometimes deformed by removing the next modality [28].

When does a specification over linearizations make sense? In fact, a specification of distributed systems given over linearisations (by means of MSO or LTL) is meaningful only if it is satisfied by *all* or *none* of the equivalent linearisations of any given distributed behaviour. We say a distributed specification is *closed* if it satisfies this inevitable semantic closure condition. In some particular cases, it is decidable to check whether a given specification, e. g., in LTL over words, is closed [29, 27]. But this does not provide a convenient specification language, which would syntactically ensure that all specifications are closed. On the other hand, logics over graph-based representations naturally eliminate this problem since the semantics is independent of the linearisations.

Why care beyond reachability? Very often people care only about reachability, and not beyond it. One of the main reasons is that, in the case of sequential non-recursive systems, the model-checking and satisfiability problems reduce to reachability. Most of the decision procedures proceed by building a machine which accepts the models of the specification. This is then followed by boolean operations on the machine model and finally performing an emptiness checking on the resulting machine which is nothing but a reachability test. However, for distributed systems, such translations from specifications to machine models do not exist. Closure under boolean operations also does not hold in general. Hence the model checking and satisfiability problems in the distributed setting cannot be reduced to reachability. Thus, while it is necessary and important to study the basic reachability problem, it is not sufficient. We need to devise techniques / verification procedures for specifications given in logical languages.

But we have techniques and tools available for words. What about graphs? In fact, graph theory is a very well-studied and mature discipline. We may use the insights and results from graph theory to our advantage. For example, generic logics on graphs serve as a good specification formalism. Graph measures such as tree-width (or clique-width or split-width) could offer good under-approximation parameters towards regaining decidability of our Turing powerful systems. The generic proof technique via tree interpretations helps us in obtaining efficient algorithms. We explore these directions in this paper with the help of split-width.

Split-width? Tree-interpretations? Split-width [16, 15, 2] offers an intuitive visual technique to decompose our behaviour graphs such as MSCs and nested words. The decomposition is mainly a divide-and-conquer technique which naturally results in a tree decomposition. Every behaviour can now be interpreted over its decomposition tree. Properties over the behaviour naturally transfer into properties over the decomposition tree. This allows us to use tree-automata techniques to obtain uniform and efficient decision procedures for a range of problems such as reachability, model checking against logical formalisms etc. Furthermore, the simple visual mechanism of split-width is as powerful as yardstick graph measures such as tree-width or clique-width. Hence it captures any class of distributed behaviours with a decidable MSO theory.

How efficient are the decision preocedures? Since graphs have a richer structure, and allow richer specifications, the verification problems are more challenging in the case of graphs as compared to words. However, our decision procedures for visual behaviours via split-width match the same time complexity as the decision procedures based on the interleaving semantics. In short, the visual technique solves more for the same price.

What you see in Table 1 is what you get. The rest of this article illustrates this.

2 Communicating Recursive Programs

We aim at analysing complex distributed systems consisting of several multithreaded recursive programs communicating via channels. In this section, we introduce the abstract model for such systems and we give its operational semantics resulting in linear behaviours. We also recall some undecidability results on these systems.

The overall structure of a system is given by its architecture, consisting of a finite set of processes, and a finite set of data structures. We are mainly interested in stack and queue data structures, though we also handle bags.

Table 1 Comparing interleavings and graphs: WYSIWYG.

WYG / WYS	Understanding (Behaviours)	Expressiveness (Specifications)	Efficiency (Complexity of algorithms)
Words	– interleaved sequence of events. Interactions are obfuscated and very difficult to recover. – combinatorial explosion (single distributed behaviour results in a huge number of interleaved traces)	– too weak for many natural specifications – requires semantical closure to be meaningful: equivalent linear traces should agree on a specification	– undecidable in general – decidable under restrictions – reductions to sequential word automata – many tools available on the shelf – good space complexity
Graphs	– visual description of events – interactions are visible / self-explained – no combinatorial explosion	– powerful specifications. trivial to express interactions – independent of particular linearisation (i. e., naturally meaningful)	– undecidable in general – decidable under (more lenient) restrictions – reductions to tree automata via tree-interpretations – good time complexity

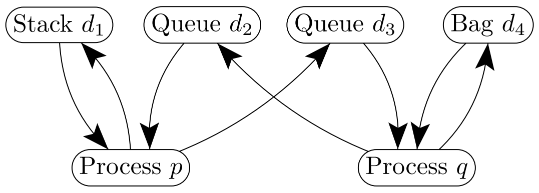


Figure 1 An Architecture. It has four data structures and two processes. Writer and Reader of the data structures are depicted by the incoming and outgoing arrows respectively.

An architecture. $\mathfrak{A}$ is a tuple $(\mathsf{Procs}, \mathsf{DS}, \mathsf{Writer}, \mathsf{Reader})$ consisting of a finite set Procs of processes, a finite set $\mathsf{DS} = \mathsf{Bags} \uplus \mathsf{Stacks} \uplus \mathsf{Queues}$ of data structures and functions Writer and Reader which assign to each data structure the process that will write into it and the process that will read from it respectively. In the special case of communicating recursive programs, we use stacks for recursion and queues for FIFO message passing. Bags are useful when no specific order is imposed on the message delivery. Since stacks are used to model recursion, we assume that $\mathsf{Writer}(s) = \mathsf{Reader}(s)$ for all $s \in \mathsf{Stacks}$. An architecture is depicted in Figure 1.

Each process is described as a finite state machine in which transitions may either be internal, only modifying the local state of the machine, or access a data structure. In the latter case, it is executing either a write event, adding a value to the data structure, or a read event, removing a value from the data structure.

Since these data structures permit only destructive reads, they induce a binary matching relation between write events and read events of a behaviour.

▶ **Definition 1.** A system of concurrent processes with data structures (CPDS) over an architecture $\mathfrak{A}$ and an alphabet Σ of action names is a tuple $\mathcal{S} = (\mathsf{Locs}, \mathsf{Val}, (\mathsf{Trans}_p)_{p\in\mathsf{Procs}}, \ell_{\text{in}}, \mathsf{Fin})$ where Locs is a finite set of locations, Val is a finite set of values that can be stored in the data structures, $\ell_{\text{in}} \in \mathsf{Locs}$ is the initial location, $\mathsf{Fin} \subseteq \mathsf{Locs}^{\mathsf{Procs}}$ is the set of global final locations, and Trans_p is the set of transitions of process p. Trans_p may have write (resp. read) transitions on data structure d only if $\mathsf{Writer}(d) = p$ (resp. $\mathsf{Reader}(p) = d$). For $\ell, \ell' \in \mathsf{Locs}$, $a \in \Sigma$, $d \in \mathsf{DS}$ and $v \in \mathsf{Val}$, Trans_p has

- internal transitions of the form $\ell \xrightarrow{a} \ell'$,
- write transitions of the form $\ell \xrightarrow{a,d!v} \ell'$ with $\mathsf{Writer}(d) = p$, and
- read transitions of the form $\ell \xrightarrow{a,d?v} \ell'$ with $\mathsf{Reader}(d) = p$.

The operational semantics of a CPDS $\mathcal{S}$ may be given as an infinite state transition system $\mathcal{TS}$. The infinite set of states of $\mathcal{TS}$ is $\mathsf{Locs}^{\mathsf{Procs}} \times (\mathsf{Val}^*)^{\mathsf{DS}}$. In the following, a state of $\mathcal{TS}$ is a pair $(\bar{\ell}, \bar{z})$ where $\bar{\ell} = (\ell_p)_{p\in\mathsf{Procs}}$ and $\bar{z} = (z_d)_{d\in\mathsf{DS}}$. Such a state is initial if $\ell_p = \ell_{\text{in}}$ for all $p \in \mathsf{Procs}$ and $z_d = \varepsilon$ for all $d \in \mathsf{DS}$. It is final if $\bar{\ell} \in \mathsf{Fin}$ and $z_d = \varepsilon$ for all $d \in \mathsf{DS}$. The transitions of the CPDS $\mathcal{S}$ induce the transitions of $\mathcal{TS}$ as follows.

- $(\bar{\ell}, \bar{z}) \xRightarrow{p,a} (\bar{\ell'}, \bar{z})$ if $\ell_p \xrightarrow{a} \ell'_p$ in Trans_p and $\ell'_q = \ell_q$ for all $q \neq p$,
- $(\bar{\ell}, \bar{z}) \xRightarrow{p,a,d!} (\bar{\ell'}, \bar{z'})$ if $\ell_p \xrightarrow{a,d!v} \ell'_p$ in Trans_p for some value $v \in \mathsf{Val}$ and $\ell'_q = \ell_q$ for all $q \neq p$, and $z'_d = z_d v$, and $z'_c = z_c$ for all $c \neq d$,
- $(\bar{\ell}, \bar{z}) \xRightarrow{p,a,d?} (\bar{\ell'}, \bar{z'})$ if $\ell_p \xrightarrow{a,d?v} \ell'_p$ in Trans_p for some value $v \in \mathsf{Val}$ and $\ell'_q = \ell_q$ for all $q \neq p$, and $z'_c = z_c$ for all $c \neq d$, and $z_d = uvw$ and $z'_d = uw$ for some $u, w \in \mathsf{Val}^*$. If $d \in \mathsf{Queues}$ (resp. $d \in \mathsf{Stacks}$) we require in addition that $u = \varepsilon$ (resp. $w = \varepsilon$).

A run of $\mathcal{TS}$ is a sequence of consecutive transitions, it is accepting if it starts in the initial state and ends in some final state of $\mathcal{TS}$. The linear trace of the run is the word obtained by concatenating the sequence of transition labels. It is a word over the alphabet $\Gamma = (\mathsf{Procs} \times \Sigma) \cup (\mathsf{Procs} \times \Sigma \times \mathsf{DS} \times \{!, ?\})$. We denote by $\mathcal{L}_{\mathsf{lin}}(\mathcal{S})$ the set of linear traces accepted by the operational semantics of $\mathcal{S}$.

Keeping the data structure access in the linear traces allows us to recover the matching relation between write actions and corresponding read actions for stacks or queues (but not for bags). This requires some counting, hence it is not a regular relation (an MSO definable relation) on the linear traces.

Undecidability. Since the operational semantics is an infinite state system, we cannot analyse it directly. The most basic problem, reachability, is already undecidable. This is in particular the case for a single process with a self queue, or a single process with two stacks, or two processes with two queues between them (the direction of the queues does not matter), or two processes having a stack each and linked with a queue (see, e. g., [15]). Characterizations of decidable topologies have been studied in the case of reliable and lossy channels [12] or in the case of FIFO channels and bags [13].

Since decidable architectures are much too restrictive, under-approximation techniques have been developped. Decidability is recovered by putting some restrictions on the possible behaviours of the system. For instance, a very natural restriction is to put a bound on the data structure capacities. The operational semantics described above becomes a finite state transition system. The analysis in this case is restricted to so-called existentially bounded behaviours [18]. Many other under-approximation classes have been studied, e. g., bounded context [30], bounded phase [21], bounded scope [23], etc.

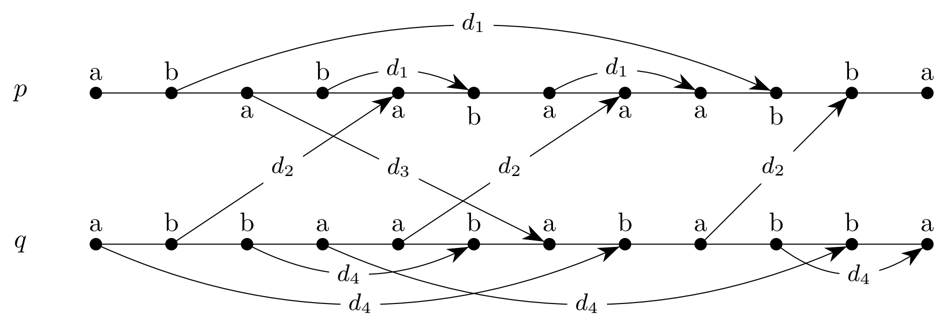


Figure 2 A CBM over the architecture given in Figure 1 and alphabet $\{a, b\}$.

3 Distributed Behaviours

In this section, we introduce the distributed semantics of CPDSs. We relate the distributed semantics and the operational semantics by showing that the linear traces arising from the latter are exactly the linearisations of the graphs defined by the former. We illustrate the benefits of considering the distributed semantics in the rest of this paper.

As a first example, the graph on Figure 2 provides a visual description of the behaviour of a CPDS. On such a graph, called concurrent behaviour with matching (CBM), the horizontal lines describe the linear behaviour of each process of the system, the other edges describe the matching relations between writes and corresponding reads. The CBM in Figure 2, is over the architecture of Figure 1. The curved arrows on process p and process q form the matching relations of stack d_1 and bag d_4 respectively. The matching relations induced by queues are shown by the arrows between the processes. As a comparison, the operational semantics generates linearisations of this behaviour such as

$$\begin{aligned}
\mathrm{Lin}_1 = {} & (p,a)(p,b,d_1,!)(p,a,d_3,!)(p,b,d_1,!)(q,a,d_4,!)(q,b,d_2,!)(q,b,d_4,!) \\
& (q,a,d_4,!)(q,a,d_2,!)(q,b,d_4,?)(q,a,d_3,?)(q,b,d_4,?)(q,a,d_2,!) \\
& (q,b,d_4,!)(q,b,d_4,?)(q,a,d_4,?)(p,a,d_2,?)(p,b,d_1,?)(p,a,d_1,!) \\
& (p,a,d_2,?)(p,a,d_1,?)(p,b,d_1,?)(p,b,d_2,?)(p,a) \\
\mathrm{Lin}_2 = {} & (p,a)(q,a,d_4,!)(p,b,d_1,!)(q,b,d_2,!)(p,a,d_3,!)(q,b,d_4,!)(p,b,d_1,!) \\
& (q,a,d_4,!)(p,b,d_1,?)(q,a,d_2,!)(p,a,d_2,?)(q,b,d_4,?)(p,a,d_1,!) \\
& (q,a,d_3,?)(p,a,d_2,?)(q,b,d_4,?)(p,a,d_1,?)(q,a,d_2,!) \\
& (p,b,d_1,?)(q,b,d_4,!)(p,b,d_2,?)(q,b,d_4,?)(p,a)(q,a,d_4,?)
\end{aligned}$$

from which it is much harder to get an intuition of the interactions going on in this behaviour. Actually, when we have bags, we cannot uniquely reconstruct a CBM from a linearisation. Hence, for distributed systems, graphs provide a visual and intuitive description of behaviours well-suited for human beings.

In the next sections, we discuss further the benefits of describing behaviours as directed graphs. Here, we define these CBMs. The intuition is easy, we have one linear trace for each process and in addition binary matching relations relating write events to corresponding reads. The formal definition has to state additional properties so that the matching relations comply with the access policies of data structures.

▶ **Definition 2.** A concurrent behaviour with matching (CBM) over architecture $\mathfrak{A}$ and alphabet Σ is a tuple $\mathcal{M} = ((w_p)_{p \in \mathsf{Procs}}, (\rhd^d)_{d \in \mathsf{DS}})$ where $w_p \in \Sigma^*$ is the sequence of events on process p and $\rhd^d$ is the binary relation matching write events on data structure d with their corresponding read events. We let $\mathcal{E}_p = \{(p,i) \mid 1 \leq i \leq |w_p|\}$ be the set of events on process $p \in \mathsf{Procs}$ and $\mathcal{E} = \bigcup_{p\in\mathsf{Procs}} \mathcal{E}_p$. For an event $e = (p,i) \in \mathcal{E}_p$, we set $\mathsf{pid}(e) = p$ and $\lambda(e)$ be the ith letter of w_p. We write $\to$ for the successor relation on processes: $(p,i) \to (p,i+1)$ if $1 \leq i < |w_p|$.

The matching relations should comply with the architecture: $\rhd^d \subseteq \mathcal{E}_{\mathsf{Writer}(d)} \times \mathcal{E}_{\mathsf{Reader}(d)}$ for all $d \in \mathsf{DS}$ and data structure accesses are disjoint: if $e_1 \rhd^d e_2$ and $e_3 \rhd^{d'} e_4$ are different edges ($d \neq d'$ or $(e_1,e_2) \neq (e_3,e_4)$) then they are disjoint ($|\{e_1,e_2,e_3,e_4\}| = 4$). Finally, writes should precede reads, so we require the relation $< = (\to \cup \rhd)^+$ to be a strict partial order on the set $\mathcal{E}$ of events, where $\rhd = \bigcup_{d\in\mathsf{DS}} \rhd^d$ is the set of all matching edges. There are no additional constraints for bags, but for stacks or queues we have to impose in addition

- $\forall d \in \mathsf{Stacks}$, $\rhd^d$ conforms to LIFO: if $e_1 \rhd^d f_1$, $e_2 \rhd^d f_2$ and $e_1 < e_2 < f_1$ then $f_2 < f_1$,
- $\forall d \in \mathsf{Queues}$, $\rhd^d$ conforms to FIFO: if $e_1 \rhd^d f_1$, $e_2 \rhd^d f_2$ and $e_1 < e_2$ then $f_1 < f_2$.

We let $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ be the set of CBMs over $\mathfrak{A}$ and Σ.

A run of a CPDS $\mathcal{S}$ on a CBM $\mathcal{M}$ is simply a labelling $\rho : \mathcal{E} \to \mathsf{Locs}$ of events by states which is compatible with the transition relation. We denote by $\rho^- : \mathcal{E} \to \mathsf{Locs}$ the map that associates with each event the state which labels its predecessor: $\rho^-(e) = \rho(e')$ if $e' \to e$ and $\rho^-(e) = \ell_{\text{in}}$ if e is minimal on its process. Then, the map ρ is a run if

(T$_1$) for all $e \rhd^d f$, there exists some value $v \in \mathsf{Val}$ such that $\rho^-(e) \xrightarrow{\lambda(e),d!v} \rho(e)$ in $\mathsf{Trans}_{\mathsf{pid}(e)}$ and $\rho^-(f) \xrightarrow{\lambda(f),d?v} \rho(f)$ in $\mathsf{Trans}_{\mathsf{pid}(f)}$,

(T$_2$) for all internal events e we have $\rho^-(e) \xrightarrow{\lambda(e)} \rho(e)$ in $\mathsf{Trans}_{\mathsf{pid}(e)}$.

A run ρ is accepting if $\mathsf{last}^\rho \in \mathsf{Fin}$ where $\mathsf{last}^\rho \in \mathsf{Locs}^{\mathsf{Procs}}$ gives the final location of the run for each process: $\mathsf{last}^\rho_p = \ell_{\text{in}}$ if $\mathcal{E}_p = \emptyset$ and $\mathsf{last}^\rho_p = \rho(\max(\mathcal{E}_p))$ otherwise. We denote by $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$ the set of CBMs accepted by $\mathcal{S}$.

We explain now the relationship between the *distributed* semantics and the sequential operational semantics. Intuitively, the linear traces of the operational semantics are precisely the linearisations of the CBMs accepted by the distributed semantics. Let us make this statement more precise. Consider a CBM $\mathcal{M}$ and define the labelling $\gamma \colon \mathcal{E} \to \Gamma$ where $\Gamma = (\mathsf{Procs} \times \Sigma) \cup (\mathsf{Procs} \times \Sigma \times \mathsf{DS} \times \{!,?\})$ by $\gamma(e) = (\mathsf{pid}(e), \lambda(e))$ if e is an internal event, and if $e \rhd^d f$ then $\gamma(e) = (\mathsf{pid}(e), \lambda(e), d!)$ and $\gamma(f) = (\mathsf{pid}(f), \lambda(f), d?)$. A linearisation of $\mathcal{M}$ is given by a total order $\sqsubseteq_{\mathsf{lin}}$ on the set $\mathcal{E}$ of events which is compatible with the causality relation: $< \subseteq \sqsubseteq_{\mathsf{lin}}$. It defines the word $\gamma(e_1)\gamma(e_2)\cdots\gamma(e_n) \in \Gamma^*$ if the linear order on $\mathcal{E}$ is $e_1 \sqsubset_{\mathsf{lin}} e_2 \sqsubset_{\mathsf{lin}} \cdots \sqsubset_{\mathsf{lin}} e_n$. We denote by $\mathsf{Lin}(\mathcal{M}) \subseteq \Gamma^*$ the set of linearisations of $\mathcal{M}$.

▶ **Theorem 3.** *We have* $\mathsf{Lin}(\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})) = \mathcal{L}_{\mathsf{lin}}(\mathcal{S})$.

However, there are also subtle differences between these two semantics. As seen above, it is possible to derive the linear traces from the CBMs, but the converse is not possible in general. For instance, if the data structure d is a bag, then the linear trace $(p, a_1, d!)(p, a_2, d!)(p, a_3, d?)(p, a_4, d?)$ is both a linearisation of the CBM $\mathcal{M}_1$ which matches a_1 with a_3 and the CBM $\mathcal{M}_2$ which matches a_1 with a_4. Hence, some specifications involving the matching relations may not be expressible on the linearisations.

If we only have stacks and queues, then we can unambiguously reconstruct a CBM from a linear trace $w \in \mathcal{L}_{\mathsf{lin}}(\mathcal{S})$. This is achieved as follows. For each $p \in \mathsf{Procs}$, the sequence of actions executed by process p is the word $w_p \in \Sigma^*$ obtained as the projection on Σ of the subword of w consisting of the letters whose first component is process p. This yields the first component $(w_p)_{p \in \mathsf{Procs}}$ of the CBM $\mathcal{M}$ associated with w. Notice that there is a natural bijection between the set of positions of w and the set $\mathcal{E}$ of events of $\mathcal{M}$. To define the matching relations, consider two events e, f associated with positions i, j of w. Then, $e \rhd^d f$ iff the letters of w at positions i and j are $w(i) \in \mathsf{Procs} \times \Sigma \times \{d!\}$ and $w(j) \in \mathsf{Procs} \times \Sigma \times \{d?\}$ and

- if $d \in \mathsf{Queues}$ then the number of writes to d before i equals the number of reads from d before j,
- if $d \in \mathsf{Stacks}$ then j is the minimal position after i such that between i and j, the number of writes to d equals the number of reads from d.

Notice that, even in the case of stacks and queues for which the matching relations can be unambiguously recovered from linear traces, these relations are not MSO definable on the linear traces. Hence, even with the powerful MSO logic, we cannot specify properties involving matching relations on linear traces. We will discuss this more precisely in the next section.

4 Specifications

The simplest specifications consist of local state reachability or global state reachability: is there a run of $\mathcal{S}$ which reaches a given local state $\ell \in \mathsf{Locs}$ on some process $p \in \mathsf{Procs}$ or a given global state $\bar{\ell} \in \mathsf{Locs}^{\mathsf{Procs}}$. To express more elaborate properties, we need some specification languages, such as first-order logic, monadic second-order logic, temporal logic, propositional dynamic logic, etc. Here it makes a big difference whether we work with the sequential operational semantics or the distributed semantics. In the first case, traces are words and the logics will refer to the linear order $\sqsubseteq_{\mathsf{lin}}$, whereas in the latter case, behaviours are graphs and the logic will have direct access to the causal ordering $<$ as well as to the process successor relation $\rightarrow$ and the matching relations $\rhd^d$. The process successor relation $\rightarrow$ can be easily recovered from the linear order $\sqsubseteq_{\mathsf{lin}}$. This is not the case for $\rhd^d$, hence also for the causal ordering $<$, in the logics mentioned above. Actually, recovering a relation $\rhd^d$ is possible if d is a stack or a queue but it requires some counting as explained above. This counting is not possible, even in the powerful MSO logic, unless the capacity of the data structure d is bounded by some fixed value B. In this case, it is possible to express $\rhd^d$ in MSO, though the formula is non-trivial and depends on the bound B. Hence, we favour specification logics on CBMs rather than on linear traces.

The partial order $< = (\rightarrow \cup \rhd)^+$ that comes with a CBM $\mathcal{M} = ((w_p)_{p \in \mathsf{Procs}}, (\rhd^d)_{d \in \mathsf{DS}})$ describes the causality relation between events. Some specifications rely on this causality relation. For instance, a distributed system may receive requests on some process p, do some internal computation involving several other processes, and finally deliver an answer on some other process q. A natural specification is that every request should be answered. Indeed, the answer to a request should be in its causal future. Such a specification is easy to write on CBMs where the causal ordering is available. For instance, it corresponds to the first order formula $\varphi = \forall x\,(\mathrm{request}(x) \implies \exists y\,(x < y \wedge \mathrm{response}(y)))$ or to the local LTL formula $\mathsf{G}(\mathrm{request} \implies \mathsf{F}\,\mathrm{response})$ (where G means *for all events in the causal future* and F means *for some event in the causal future*). The CBM $\mathcal{M}$ depicted in Figure 3 does not satisfy this specification as Request 2 is not responded. Request 1 has Response 2 in its future, though not Response 1. Recall that, by Theorem 3, linear traces are linear extensions of CBMs and events that are concurent in $\mathcal{M}$ may be ordered arbitrarily in a linear trace. Therefore, $\mathsf{Lin}(\mathcal{M})$ includes many linearisations in which the responses follow the requests and from which it is not easy to see whether the specification is satisfied or not.

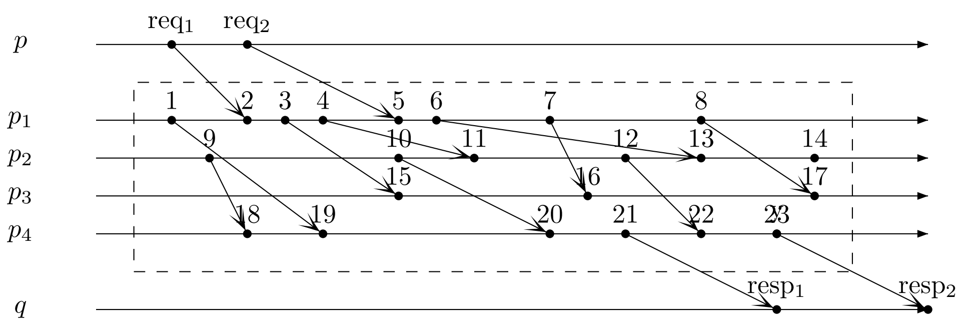


Figure 3 A request/response scenario.

As explained above, recovering the causal order $<$ from the total order $\sqsubseteq_{\mathsf{lin}}$ is not possible, even with very expressive specification languages such as MSO over words. As a conclusion, a simple and natural specification such as the request/response property, cannot be reduced to a reachability problem on the operational semantics in general. Such a reduction is possible when the data structures are restricted to bounded stacks and bounded queues (no bags), but it is non-trivial.

The same argument holds for specifications that involve the matching relation associated with a stack. For instance, we may specify that after receiving a request, the process calls a recursive procedure and when *this* call returns it *immediately* delivers the response. Again, matching a call with the corresponding return requires counting which is not a regular property unless the call depth is bounded.

As another example, a specification may require that when an access to some critical section is denied, there is a good reason for that, say some concurrent event is accessing the critical section. Again, concurrency – which is the absence of causal ordering – cannot be expressed on the linear traces in general.

We introduce below two powerful specification languages on CBMs. First, monadic second-order logic over $\mathsf{CBM}(\mathfrak{A},\Sigma)$ is denoted $\mathsf{MSO}(\mathfrak{A},\Sigma)$. It follows the syntax:

$$\varphi ::= \mathsf{false} \mid a(x) \mid p(x) \mid x \leq y \mid x \rhd^d y \mid x \to y \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x\, \varphi \mid \exists X\, \varphi$$

where $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$ and $a \in \Sigma$. The semantics is as expected. Every sentence φ in MSO defines a language $\mathcal{L}_{\mathsf{cbm}}(\varphi) \subseteq \mathsf{CBM}(\mathfrak{A},\Sigma)$ consisting of all CBMs that satisfy that sentence. A language $L \subseteq \mathsf{CBM}(\mathfrak{A},\Sigma)$ is MSO definable if $L = \mathcal{L}_{\mathsf{cbm}}(\varphi)$ for some sentence $\varphi \in \mathsf{MSO}(\mathfrak{A},\Sigma)$.

▶ Remark. The set $\mathsf{CBM}(\mathfrak{A},\Sigma)$ is MSO definable in the class of graphs over the signature associated with $(\mathfrak{A},\Sigma)$. More precisely, there is an $\mathsf{MSO}(\mathfrak{A},\Sigma)$ sentence Φ_{cbm} such that a graph $G = (\mathcal{E}, \to, (\rhd^d)_{d\in\mathsf{DS}}, \mathsf{pid}, \lambda)$ satisfies Φ_{cbm} iff it is a CBM over $(\mathfrak{A},\Sigma)$. It is easy to obtain the formula Φ_{cbm} from Definition 2, including the LIFO and FIFO conditions for stacks and queues.

▶ Remark. The language $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$ of a CPDS $\mathcal{S}$ is definable with an existential MSO sentence $\Phi_{\mathcal{S}}$. Intuitively, with an existential prefix $\exists(X_{p,\tau})_{p\in\mathsf{Procs},\tau\in\mathsf{Trans}_p}$ the formula guesses for each transition $\tau \in \mathsf{Trans}_p$ the set of events from process $p \in \mathsf{Procs}$ that will execute this transition, and then checks with a first-order formula that this guess defines an accepting run of $\mathcal{S}$.

▶ Remark. Notice that CBM-graphs have degree bounded by 3 since any event may take part in at most one matching relation. Therefore, on CBMs, the logic MSO_2 in which we may also quantify over edges (individual variables or set variables) has the same expressive power as MSO.

We are interested in two decision problems: satisfiability of a specification and model checking of a system against a specification. Given an architecture $\mathfrak{A}$, an alphabet Σ and an MSO sentence $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$, the satisfiability problem asks whether $\mathcal{M} \models \varphi$ for some $\mathcal{M} \in \mathsf{CBM}(\mathfrak{A}, \Sigma)$. For the model checking problem, we are also given a CPDS $\mathcal{S}$ and we ask whether the specification is satisfied for all (or for some) behaviours of the system: $\mathcal{M} \models \varphi$ for all $\mathcal{M} \in \mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$.

Since reachability (or equivalently emptiness) is undecidable in general for CPDSs, both satisfiability and model checking are undecidable for any specification language that can express reachability, in particular MSO. This is trivial for model checking: the specification false is satisfied iff the language of $\mathcal{S}$ is empty, i. e., if the final states are not reachable. For satisfiability, it follows from the remark above since $\Phi_{\mathcal{S}}$ is satisfiable iff the set of final states is reachable in $\mathcal{S}$.

▶ Remark. We have seen above that reachability reduces to model-checking or to satisfiability. In the case of finite sequential systems, the converse holds since, for any MSO formula φ, we can compute an automaton $\mathcal{A}_\varphi$ which accepts exactly the models of φ [11, 17, 32]. Then, the model-checking problem reduces to the emptiness problem for the intersection of the system and the negation of the formula. But this approach fails for distributed systems because it is not possible in general to compute an automaton equivalent to a given formula. Indeed, we have already seen that the matching relation, hence also the partial order, cannot be computed by an automaton on the linearisations. Even if we stay in the distributed semantics, an MSO formula cannot be translated to a CPDS in general. This is because even the simpler class of message passing automata (i. e., when DS = Queues) is not closed under complementation [9]. Therefore the model-checking problem for CPDSs and MSO does not reduce to reachability in general.

Is MSO the ultimate logic? MSO is a very expressive logic for specifications. Its drawback is that, even when we recover decidability by restricting to some under-approximation class, the complexity of the decision procedure in non-elementary. This is already the case for words or trees. To get better complexity, one should use other formalisms such as Temporal Logics or Propositional Dynamic Logic (PDL). Towards expressing properties of CBMs, the classical LTL over words has been extended to *visible* behaviours such as nested words [4, 3], MSCs [8], nested traces [7], multiply nested words [24], etc. These temporal logics have explicit modalities that allow one to retrieve matching edges or to follow the partial order. Below, we describe propositional dynamic logic which embeds all these logics and provides powerful navigational abilities.

In PDL, there are two types of formulas. State formulas (σ) describe the properties of events in a behaviours, hence they have an implicit first-order free variable assigned to the current event. Atomic propositions such as p or a assert that the current event is on process $p \in \mathsf{Procs}$ or is labelled $a \in \Sigma$. In addition to boolean connectives, we have a path modality $\langle\pi\rangle\sigma$ claiming the existence of a path following π from the current node to an event satisfying σ. A path formula π has two implicit first-order free variables assigned to the end points of the path. They are built from basic moves following edges of the graph (in our case $\rightarrow$ and $\rhd^d$), either forwards or backwards, using rational expressions that may use intersection in addition to the classical union, concatenation and iteration. In addition, we may check a state formula σ along a path. Formally, the syntax of $\mathsf{ICPDL}(\mathfrak{A}, \Sigma)$ is given by

$$\sigma ::= \mathsf{false} \mid p \mid a \mid \sigma \vee \sigma \mid \neg\sigma \mid \langle\pi\rangle\sigma$$

$$\pi ::= \mathsf{test}(\sigma) \mid \rightarrow \mid \rhd^d \mid \pi^{-1} \mid \pi + \pi \mid \pi \cap \pi \mid \pi \cdot \pi \mid \pi^*$$

where $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$ and $a \in \Sigma$. If intersection $\pi \cap \pi$ is not allowed, the fragment is PDL with converse (CPDL)[1].

5 Graph theoretic approach to verification

In this section, we show how results from graph theory may help in designing decidable under-approximation techniques for the verification of CPDSs. The distributed semantics defines the behaviours as graphs, hence we are interested in checking properties of the set of graphs $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$ accepted by a CPDS. More precisely, our aim is to solve the model checking problem: given a system $\mathcal{S}$ and a specification $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$, does $\mathcal{S} \models \varphi$, i. e., is the formula φ valid on $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$?

Let us fix some architecture $\mathfrak{A}$ and the set Σ of action labels. Decidability of the model-checking problem is equivalent to decidability of the MSO theory of the set $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ of CBM-graphs. Indeed, we have seen in Remark 4 that from a CPDS $\mathcal{S}$ we can compute a formula $\Phi_{\mathcal{S}}$ which defines $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$. Therefore, $\mathcal{S} \models \varphi$ iff $\neg\Phi_{\mathcal{S}} \vee \varphi$ is valid on $\mathsf{CBM}(\mathfrak{A}, \Sigma)$. Hence, decidability of model-checking reduces to decidability of the MSO theory of $\mathsf{CBM}(\mathfrak{A}, \Sigma)$. For the converse, it suffices to consider a universal CPDS $\mathcal{S}$ with $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) = \mathsf{CBM}(\mathfrak{A}, \Sigma)$.

We have seen in Section 2 that reachability, the most basic model-checking problem, is undecidable for CPDS, even for very simple architectures such as two processes communicating via FIFO channels (with no stacks) or a single process with two stacks. Hence, the MSO theory of $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ is undecidable in general, which can be seen also directly since CBMs have unbounded tree-width (or clique-width) in general. Still, it is extremely challenging to develop correct programs for distributed multi-threaded recursive systems. Hence, techniques for approximate verification have been extensively developed recently. We will not discuss over-approximation techniques in the present paper.

An under-approximation technique restricts the problems (reachability, satisfiability or model-checking) to a decidable subclass $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ of behaviours. Often the subclass $\mathcal{C}_m$ is parametrised with some integer m. We cover more behaviours by increasing the parameter m. The approximation family $(\mathcal{C}_m)_{m\geq 0}$ is *complete* or *exhaustive* if $\mathsf{CBM}(\mathfrak{A}, \Sigma) = \bigcup_m \mathcal{C}_m$. Hence, the aim of under-approximate verification is to define and study *meaningful* classes $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ with a decidable MSO theory.

Decidability of the MSO theory (or equivalently decidability of the MSO satisfiabiliy problem) for classes $\mathcal{C}$ of graphs has been extensively studied. We recall now some important results that will be useful for our purpose (see [14, Chapter 1] for a survey). Recall that CBM-graphs have degree bounded by 3 since any event may take part in at most one matching relation. Hence, we restrict our attention to results for classes $\mathcal{C}$ of bounded degree graphs. The following fact summarizes some of the main results (see Theorem 4).

An MSO definable class $\mathcal{C}$ of bounded degree graphs has a decidable MSO theory iff it can be interpreted[2] in the class of binary (labelled) trees.

[1] If backward paths π^{-1} are not allowed the fragment is called PDL with intersection (IPDL). In simple PDL neither backwards paths nor intersection is allowed.

[2] There are several equivalent ways to define an interpretation of a graph $G = (V, E)$ in a labelled tree T. We describe the MSO-transductions of Courcelle, but one may, for example, also use the regular path descriptions of Engelfriet and van Oostrom. An MSO-interpretation is given by a tuple of MSO formulas. We will give a concrete example in Section 6. Intuitively, not all labelled trees admit a valid graph interpretation, hence, we use a sentence Φ_{valid} to select the "good" trees. The vertices of the graph are some nodes of the tree, and we use a formula $\Phi_{\mathsf{vertex}}(x)$ to select those nodes of T which should be interpreted as vertices of G: $V = \{u \in T \mid T \models \Phi_{\mathsf{vertex}}(u)\}$. Finally, a formula $\Phi_{\mathsf{edge}}(x, y)$ encodes

In the light of this fact, under-approximation classes are obtained by MSO definitions together with tree interpretations. Then, verification problems are reduced to problems on tree automata, yielding efficient algorithms.

Such tree interpretations can be defined specifically for some class $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$. This is for instance the case for bounded phase behaviours of multi-pushdown automata [21] where multiply nested words of bounded phase are interpreted in binary trees called stack-trees. Another example is given by the interpretation of some classes of multiply nested words in visibly (k-)path trees in order to prove decidability of emptiness and closure under complement of multi-pushdown automata when restricted to some classes of behaviours that can be interpreted in these path-trees [25] .

A higher level approach is to prove some combinatorial property on the class $\mathcal{C}$ which ensures the existence of a tree interpretation. For instance, one may show that the class $\mathcal{C}$ has bounded tree-width (and is MSO definable). This is the approach taken in [26], where decidability of several under-approximation classes is established by proving that they are MSO definable and have bounded tree-width. For most of the classes considered in [26] the decidability had been already proved directly. Hence, [26] provides a unifying approach as well as efficient algorithms based on tree interpretations.

Alternative combinatorial properties may be more convenient, for instance bounded clique-width, which is equivalent to bounded tree-width on classes of bounded degree graphs. In [16, 2] another decomposition technique, called split-width, is defined specifically for CBMs (see Section 6). On classes of CBMs, bounded tree-width, bounded clique-width and bounded split-width are all equivalent. We believe that, for a class of CBMs, establishing a bound on split-width is easier than the other measures. Also, we will see in Section 6 that split-width gives an easy and natural interpretation of CBMs in binary trees. Hence, split-width provides a convenient, necessary and sufficient condition, to establish decidability of the MSO theory of an under-approximation class.

The following theorem summarizes some of the relevant results. For more details, the reader is referred to [14, Chapter 1] and [16, 15].

▶ **Theorem 4.** *Let $\mathcal{C}$ be a class of bounded degree graphs which is* MSO *definable. TFAE*
1. *$\mathcal{C}$ has a decidable* MSO *theory,*
2. *$\mathcal{C}$ can be interpreted in binary trees,*
3. *$\mathcal{C}$ has bounded tree-width,*
4. *$\mathcal{C}$ has bounded clique-width,*
5. *$\mathcal{C}$ has bounded split-width (if $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ is a class of* CBMs*).*

In order to define a good under-approximation class, one may show that it is MSO definable and that it satisfies one of the conditions of Theorem 4. We will introduce split-width in the next section and show that it is a convenient tools for CBMs. Proving MSO definability is often easy. This is the case for many under-approximation classes, like bounded context, bounded phase, bounded scope, ordered etc. for multi-pushdown systems. Also, for distributed systems, it is easy to give an MSO definition for universally bounded MSCs, or the bounded context and well-queuing assumption of [22, 19].

the edge relation of G: $T \models \Phi_{\mathsf{edge}}(u, v)$ iff $(u, v) \in E$. We may also interpret vertex-labelled graphs by refining Φ_{vertex} in a tuple of formulæ $\Phi_a(x)$ which selects those vertices/nodes that are labelled a. Finally, in case of edge-labelled graphs, the formula $\Phi_{\mathsf{edge}}(x, y)$ is refined in a tuple of formulæ $\Phi_d(x, y)$ one for each edge-label d.

6 Split-width

In this section, we introduce split-width. We explain the associated tree-interpretations and infer the decidability and complexity of a collection of verification problems when parametrised by split-width. We also discuss how to use split-width in order to obtain similar results for various under-approximation classes.

With K. Narayan Kumar, we introduced split-width in [16] for multiply nested words. The technique was later extended to CBMs in [15, 2].

The idea is to decompose a graph in atomic pieces consisting of matching write/read pairs, see Figure 4. This can be seen as a two-player turn-based game with a fixed budget k which will be the width of the decomposition. The existential player (Eve), trying to prove the existence of a decomposition of width at most k, has to disconnect the CBM graph by splitting at most k process edges. For instance, the root of Figure 4 is labelled with a CBM $\mathcal{M}$ over the architecture $\mathfrak{A}$ of Figure 1. The graph $\mathcal{M}$ cannot be disconnected by splitting only one or two process edges. So Eve splits three process edges that are shown as dashed red edges in the split-CBM $\mathcal{M}'$. The universal player (Adam) will now choose one of the connected components of $\mathcal{M}'$ and the game continues. $\mathcal{M}'$ has two connected components $\mathcal{M}_1$ and $\mathcal{M}_2$, providing two choices for Adam.

If $\mathcal{M}_2$ is chosen, Eve splits the two process edges and the resulting graph $\mathcal{M}'_2$ has now two connected components. Whichever is chosen by Adam is an atomic write/read edge, which is a winning position for Eve.

Assume now that $\mathcal{M}_1$ is chosen. Note that $\mathcal{M}_1$ has two *blocks* of events on process q with one *hole* between them. The first block consists of a single event labelled b and the second one consists of three events labelled cdc. A *block* of events in a split-CBM is a maximal sequence of events on a single process. For instance, $\mathcal{M}'$ has two blocks on process p and three blocks on process q. Clearly, the number of holes on some process is the maximum of zero and the number of blocks minus one on that process. The budget k of Eve is reduced by the number of holes. For instance, $\mathcal{M}_1$ has one hole (on process q) hence Eve is only allowed two $(3-1)$ more splits to disconnect $\mathcal{M}_1$ without exceeding her budget. Her choice is depicted in $\mathcal{M}'_1$. One connected component of $\mathcal{M}'_1$ is a matching edge which is winning for Eve. So Adam should choose $\mathcal{M}_3$ lest he lose immediately. Eve splits the remaining process edge and wins regardless of Adam's choice.

To summarize, Eve wins a play if it ends in an atomic CBM, i. e., a single internal event or a matching write/read edge. She loses if she cannot disconnect a non-atomic graph without introducing more than k split-edges (holes). The split-width of a CBM is the minimum budget k for which Eve has a winning strategy. A winning strategy for Eve with budget 3 is depicted in Figure 4 for the CBM $\mathcal{M}$ at the root. As explained above, $\mathcal{M}$ cannot be disconnected with only two splits, hence its split-width is exactly 3. We denote by $\mathsf{CBM}^k_{\mathsf{split}}(\mathfrak{A}, \Sigma)$ the set of CBMs over $\mathfrak{A}$ and Σ with split-width bounded by k.

▶ **Example 5.** Nested words have split-width at most 2. Nested words [5] are CBMs over an architecture with a single process and a single stack. The bound on split-width can be seen easily since a nested word w is (a) either the concatenation of two nested words in which case Eve splits the edge between the two nested words, (b) or is of the form $\begin{tikzcd} a \arrow[r] \arrow[rr, bend left] & w \arrow[r] & b \end{tikzcd}$ where w is a nested word, in which case Eve splits the first and last process edges, (c) or is an atomic CBM.

▶ **Example 6.** Existentially k-bounded CBMs have split-width at most $k+1$. A CBM $\mathcal{M}$ is existentially k-bounded if it admits a linearization such that the number of unmatched

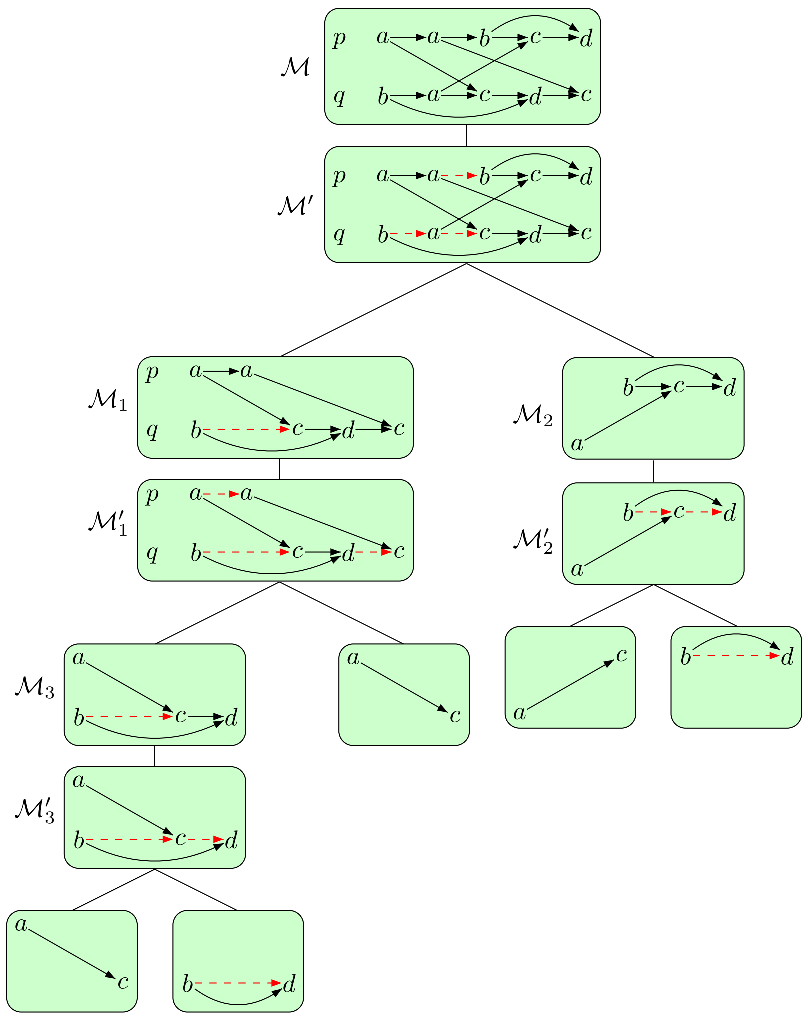


Figure 4 A split decomposition of width 3.

writes at any point is bounded by k. For instance, the CBM $\mathcal{M}$ at the root of Figure 4 is existentially 3-bounded. Let the linear order witnessing the existential bound be $\sqsubseteq_{\mathsf{lin}}$. The strategy of Eve is to detach the first $k+1$ events of $\mathcal{M}$ with respect to $\sqsubseteq_{\mathsf{lin}}$ by splitting the corresponding outgoing process edges. The resulting split-CBM $\mathcal{M}'$ must be disconnected. Indeed, the detached events cannot all be write events, otherwise the bound k is exceeded. If a read event is detached, then the corresponding write is also detached since it must come earlier in any linearization. Therefore, the split-CBM $\mathcal{M}'$ contains some connected components which are atomic CBMs and at most one connected component $\mathcal{M}_1$ which is non-atomic. Adam must choose the component $\mathcal{M}_1$ to avoid losing the game immediately. Then, Eve proceeds by splitting some more process edges until $k+1$ events are detached in the $\sqsubseteq_{\mathsf{lin}}$ order. As above, the resulting split-CBM must be disconnected and at most one of its connected component is non-atomic. Eve applies the same strategy as long as there is a non-atomic connected component.

▶ **Example 7.** Multiply nested words with at most m phases have split-width at most 2^m. Multiply nested words (MNWs) are CBMs over an architecture with a single process and several stacks. A *phase* in a MNW is a factor in which all read events are from the same stack.

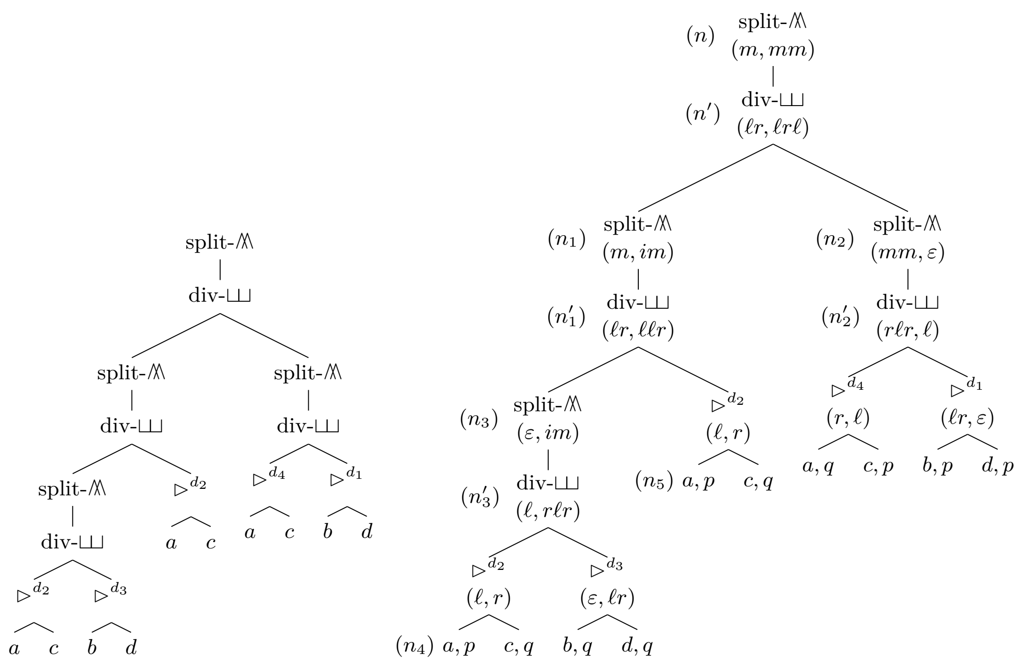


Figure 5 A split term s (left) and a labelled term t (right) corresponding to Figure 4.

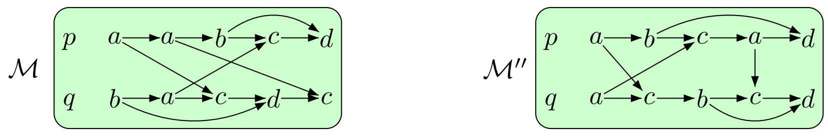


Figure 6 Two CBMs that can be decomposed with the split-term s of Figure 4.

A MNW is m-phase bounded if it is the concatenation of at most m phases [21]. All m-phase bounded MNWs have split-width at most 2^m [16]. The bounded phase under-approximation has been extended to distributed systems in [15, 1].

In fact, an upper bound on split-width has been established for many under-approximation classes. See [16] for many classes of multi-pushdown systems such as bounded scope [23], ordered [10, 6], etc. For many classes of communicating (multi-pushdown) systems, see [15, 1, 2].

Split-algebra. The split-game introduced above gives the decomposition view (top-down) which is useful to establish a bound on split-width. A winning strategy of Eve for some CBM $\mathcal{M}$ can be represented with a tree as in Figure 4. Dually, there is also a split-algebra which constructs CBMs in a bottom-up fashion starting from atomic ones using two operations: shuffle (opposite of divide) and merge (opposite of split). The terms of the split-algebra over $\mathfrak{A}$ and Σ follow the syntax:

$$s ::= a \mid a \rhd^d b \mid ⩕(s) \mid s \sqcup\!\sqcup s$$

with $a, b \in \Sigma$ and $d \in \mathsf{DS}$. The split-term s corresponding to Figure 4 is given on the left of Figure 5 (recall that the architecture is taken from Figure 1).

Several CBMs may admit a decomposition via the same split-term. For instance, the split-term s on the left of Figure 5 allows us to decompose both the CBMs $\mathcal{M}$ and $\mathcal{M}''$ of Figure 6. The main reason is that a shuffle node does not specify how the blocks of the two children are shuffled, and a merge node does not specify which holes of the child are mended into process edges. This ambiguity can be removed with an extra labelling as shown on tree t on the right of Figure 5 corresponding to the decomposition of Figure 4. At a shuffle node, the labelling consists of a tuple of words $(w_p)_{p\in\mathsf{Procs}}$, where $w_p \in \{\ell, r\}^*$ describes how the blocks on process p of the children are shuffled. For instance, the shuffle node (n') of t is labelled $(w_p, w_q) = (\ell r, \ell r \ell)$ which means that the first block of $\mathcal{M}'$ – corresponding to node (n') – on process p comes from the left child $\mathcal{M}_1$ – corresponding to (n_1) – and the second block comes from the right child $\mathcal{M}_2$ – corresponding to (n_2). On process q there are 3 blocks, first and third coming from $\mathcal{M}_1$ and second coming from $\mathcal{M}_2$. The same kind of labelling is used at $\rhd$-nodes. Now, for a merge node, the labelling is also a tuple of words $(w_p)_{p\in\mathsf{Procs}}$, but now a word $w_p \in \{i, m\}^*$ tells whether the holes (split edges) of the child are kept as such (i for inherited) or are turned into process edges (m for mended). For instance, node (n_1) – corresponding to $\mathcal{M}_1$ – is labelled (m, im) which means that from its child $\mathcal{M}'_1$ – corresponding to (n'_1) – the hole of process p is mended, the first split edge of q is inherited and the second one is mended. Also, each leaf is labelled with its corresponding process.

Note that, the width of a decomposition can be recovered from the labelled split-term (but not from the unlabelled split-term). Indeed, the labelling of a merge node directly gives the number of holes of its child, and the labelling of a shuffle node, or a $\rhd$-node, gives the number of blocks from which we can infer the number of holes.

Tree-interpretations. Each CBM that can be decomposed with a split-term s admits an MSO-interpretation (cf. Footnote 2) in the tree s, which is defined by a tuple of formulas over binary trees $(\Phi_{\mathsf{valid}}, (\Phi_a)_{a\in\Sigma}, (\Phi_p)_{p\in\mathsf{Procs}}, (\Phi_d)_{d\in\mathsf{DS}}, \Phi_{\rightarrow})$. The interpretation guesses (with set variables) a labelling to disambiguate the split term as explained above. Not all labelled split-terms allow an interpretation, so we use a formula Φ_{valid} to check the validity of the labelling. Essentially, we have to check that, for each process p, the number of blocks at a node is compatible with that of the children. For instance, a label $w_p \in \{\ell, r\}^*$ at a shuffle node assumes $|w_p|_\ell$ (resp. $|w_p|_r$) blocks on process p from the left (resp. right) child. A label $w_p \in \{i, m\}^*$ at a merge node assumes $|w_p|$ holes on process p from the child. For a $\rhd^d$ node with $p = \mathsf{Writer}(d)$ and $q = \mathsf{Reader}(d)$, we request that the children are leaves labelled p (left) and q (right), and if $p = q$ then we request $w_p = \ell r$ and $w_s = \varepsilon$ for $s \neq p$, and if $p \neq q$ then we request $w_p = \ell$, $w_q = r$ and $w_s = \varepsilon$ for $p \neq s \neq q$. In addition, for stack or queue data structures, the formula Φ_{valid} has to check that the LIFO or FIFO conditions are respected. To do so, we need to enrich further the labelling. If $d \in \mathsf{Stacks}$ then we maintain the $\rhd^d$ relation between blocks of process $p = \mathsf{Reader}(d) = \mathsf{Writer}(d)$: $i \rhd^d j$ if $e \rhd^d f$ for some e in the ith block of process p and some f in the jth block of process p. This information can be easily computed by a deterministic bottom-up tree automaton. At a shuffle node, we make sure that the LIFO condition is respected by rejecting shuffles that would result in a $\rhd^d$ relation between blocks that is not well-nested. Hence we obtain an EMSO formula to check the LIFO condition for data structure d. We proceed similarly for queues.

We denote by $\mathsf{DST}^k_{\mathsf{valid}}$ the set of split-terms *disambiguated* by a *valid* labelling of width at most k. Each tree $t \in \mathsf{DST}^k_{\mathsf{valid}}$ encodes a unique CBM of split-width at most k, denoted $\mathsf{cbm}(t)$. Conversely, every $\mathcal{M} \in \mathsf{CBM}^k_{\mathsf{split}}$ is encoded by some, often many, trees $t \in \mathsf{DST}^k_{\mathsf{valid}}$.

Vertices of $\mathsf{cbm}(t)$ are leaves of t hence we let $\Phi_{\mathsf{vertex}}(x) = \mathsf{leaf}(x)$. The vertex labelling in $\mathsf{cbm}(t)$ is the corresponding leaf labelling in t. Hence, formulæ $\Phi_a(x)$ and $\Phi_p(x)$ state that leaf x is labelled a and p in t. The matching relation also admits a trivial interpretation: for $d \in \mathsf{DS}$, $\Phi_d(x, y)$ states that x and y are leaves with a comon father labelled $\rhd^d$.

The process relation is slightly harder to recover. This is where the additional labelling is needed. Intuitively, $\Phi_\rightarrow(x, y)$ states that from leaf x it is possible to walk up the tree to some merge node m, then walk down the tree to leaf y, and that the split edge from x to y has been mended at node m. It is easy to check this property with a tree automaton and to deduce the (EMSO) formula $\Phi_\rightarrow(x, y)$. More precisely, the deterministic bottom-up tree automaton keeps in its states the block B_x of which x is the right-most event, and the block B_y of which y is the left-most event. It goes to an accepting state only if the hole between B_x and B_y is mended into a process edge at some merge node. For instance, the process edge from leaf (n_4) to leaf (n_5) is established at merge node (n_1).

Tree-width, clique-width and split-width. On CBMs, split-width is a measure that is very similar to clique-width or tree-width. It is shown in [16, 15] that for CBMs, a bound on split-width implies a (linear) bound on clique-width or tree-width and vice versa. More precisely, if a CBM has split-width k then it has clique-width at most $2(k + |\mathsf{Procs}|) + 1$ and tree-width at most $2(k + |\mathsf{Procs}|) - 1$. Conversely, if a CBM has clique-width c or tree-width t then it has split-width bounded by $2c - 3$ or $120(t + 1)$.

Verification procedures for bounded split-width. Most verification problems become decidable with reasonable complexity when parametrised by a bound on split-width. Intuitively, the tree-interpretation provided by split-width allows us to uniformly reduce a collection of problems on CBMs of bounded split-width to problems on trees, which are then solved with tree automata techniques.

More precisely, let $\mathcal{S}$ be a CPDS over $(\mathfrak{A}, \Sigma)$ and $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ be a specification. The model checking problem restricted to the class $\mathsf{CBM}^k_{\mathsf{split}}$ of CBMs with split-width bounded by k asks whether $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) \cap \mathsf{CBM}^k_{\mathsf{split}} \subseteq \mathcal{L}_{\mathsf{cbm}}(\varphi)$. Similarly, the emptiness problem for $\mathcal{S}$ (resp. the satisfiability problem for φ) restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ asks whether $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) \cap \mathsf{CBM}^k_{\mathsf{split}} = \emptyset$ (resp. $\mathcal{L}_{\mathsf{cbm}}(\varphi) \cap \mathsf{CBM}^k_{\mathsf{split}} \neq \emptyset$). We reduce these problems to the emptiness problem for tree automata as follows.

First, we can build a tree automaton $\mathcal{A}^k_{\mathsf{valid}}$ of size $2^{\mathcal{O}(k^2|\mathfrak{A}|)}$ which accepts $\mathsf{DST}^k_{\mathsf{valid}}$. Next, we can build a tree automaton $\mathcal{A}^k_{\mathcal{S}}$ of size $|\mathcal{S}|^{\mathcal{O}(k+|\mathsf{Procs}|)}$ which accepts a tree $t \in \mathsf{DST}^k_{\mathsf{valid}}$ if and only if $\mathsf{cbm}(t) \in \mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$. Therefore, the emptiness problem for the CPDS $\mathcal{S}$ restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ reduces to the emptiness problem of the tree automaton $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{S}}$.

Now, let φ be a sentence in $\mathsf{MSO}(\mathfrak{A}, \Sigma)$. Using the MSO interpretation $(\Phi_{\mathsf{valid}}, \Phi_{\mathsf{vertex}}, (\Phi_a)_{a\in\Sigma}, (\Phi_p)_{p\in\mathsf{Procs}}, (\Phi_d)_{d\in\mathsf{DS}}, \Phi_\rightarrow)$ for k-bounded split-width, we can construct a formula $\overline{\varphi}^k$ from φ such that for all trees $t \in \mathsf{DST}^k_{\mathsf{valid}}$, we have $t \models \overline{\varphi}^k$ if and only if $\mathsf{cbm}(t) \models \varphi$. By [31], from the MSO formula $\overline{\varphi}^k$ we can construct an equivalent tree automaton $\mathcal{A}^k_\varphi$. Therefore, the satisfiability problem for the MSO formula φ restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ reduces to the emptiness problem of the tree automaton $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_\varphi$.

Finally, we deduce easily that $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) \cap \mathsf{CBM}^k_{\mathsf{split}} \subseteq \mathcal{L}_{\mathsf{cbm}}(\varphi)$ if and only if $t \models \overline{\varphi}^k$ for all trees t accepted by $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{S}}$. Therefore, the model checking problem $\mathcal{S} \models \varphi$ restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ reduces to the emptiness problem for the tree automaton $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{S}} \cap \mathcal{A}^k_{\neg\varphi}$.

We have described above uniform decision procedures for an array of verification problems. We refer to [16, 15, 2] for more details and we summarise the computational complexities of these procedures in Table 2.

Table 2 Summary of the complexities for bounded split-width verification.

Problem	Complexity	
	Architecture $\mathfrak{A}$, alphabet Σ, bound k on split-width	
	being part of the input (k in unary)	being fixed
CPDS emptiness	ExpTime-Complete	PTime-Complete
CPDS inclusion or universality	2ExpTime	ExpTime-Complete
LTL/CPDL satisfiability, model checking	ExpTime-Complete	
ICPDL satisfiability or model checking	2ExpTime -Complete	
MSO satisfiability or model checking	Non-elementary	

Verification procedures for other under-approximation classes. Our approach is generic in yet another sense. Under-approximation classes which admit a bound on split-width also may benefit from the uniform decision procedures described above, provided these classes correspond to regular sets of split-terms.

More precisely, let $\mathcal{C}_m$ be an under-approximation class with $\mathcal{C}_m \subseteq \mathsf{CBM}^k_{\mathsf{split}}$. For instance, we have seen that existentially m-bounded CBMs have split-width at most $k = m + 1$ (Ex. 6) and m-bounded phase MNWs have split-width at most $k = 2^m$ (Ex. 7). Assume that we can construct[3] a tree automaton $\mathcal{A}^k_{\mathcal{C}_m}$ which accepts a tree $t \in \mathsf{DST}^k_{\mathsf{valid}}$ if and only if $\mathsf{cbm}(t) \in \mathcal{C}_m$. Then, the decision procedures can be restricted to the class $\mathcal{C}_m$ with a further intersection with the tree automaton $\mathcal{A}^k_{\mathcal{C}_m}$. For instance, the emptiness problem for $\mathcal{S}$ restricted to $\mathcal{C}_m$ reduces to the emptiness problem of $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{C}_m} \cap \mathcal{A}^k_{\mathcal{S}}$. The model checking problem $\mathcal{S} \models \varphi$ restricted to $\mathcal{C}_m$ reduces to the emptiness problem of $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{C}_m} \cap \mathcal{A}^k_{\mathcal{S}} \cap \mathcal{A}^k_{\neg\varphi}$.

Clearly, the bound k on split-width in terms of m as well as the size of $\mathcal{A}^k_{\mathcal{C}_m}$ will impact on the complexity of the decision procedures. We give below several examples.

First, nested words have split-width bounded by a constant 2, and the set of nested words can be recognised by a trivial 1-state CPDS. Hence the complexities of various problems follow the right-most column of Table 2. Notice that already for this simple case, the complexities match the corresponding lower bounds for all problems.

Next, suppose a class $\mathcal{C}_m$ admits a bound on split-width $k = \mathsf{poly}(m)$ and $\mathcal{A}^k_{\mathcal{C}_m}$ is of size[4] bounded by $2^{\mathsf{poly}(m)}$. Then the decision procedures for various problems with respect to the under-approximation class $\mathcal{C}_m$ follow the complexities given in Table 2.

This can be extended as follows. Assume the bound k on split-width of the under-approximation class $\mathcal{C}_m$ is n-fold exponential in m and that the size of the tree automaton $\mathcal{A}^k_{\mathcal{C}_m}$ is bounded by $(n + 1)$-fold exponential in m (e. g., if we have a CPDS $\mathcal{S}_m$ of size $2^{\mathsf{poly}(k)}$), then the complexities given in Table 2 (left column) will be augmented by a n-fold exponentiation. For instance, the class $\mathcal{C}_m$ of m-bounded phase MNWs has split-width bounded by 2^m (Ex. 7). Also, it is trivial to get a CPDS $\mathcal{S}_m$ for $\mathcal{C}_m$ of size $\mathsf{poly}(m)$. Hence, the size of $\mathcal{A}^k_{\mathcal{C}_m} = \mathcal{A}^k_{\mathcal{S}_m}$ is $2^{\mathsf{poly}(m)}$. We deduce that the complexities given in Table 2 (left column) are augmented by one exponentiation for m-bounded phase MNWs.

Thus the verification method via split-width is uniform not only for a wide range of

[3] One way to obtain $\mathcal{A}^k_{\mathcal{C}_m}$ is to provide a CPDS $\mathcal{S}_m$ which accepts the class $\mathcal{C}_m$, then the automaton $\mathcal{A}^k_{\mathcal{S}_m}$ serves as $\mathcal{A}^k_{\mathcal{C}_m}$. Similarly, if there is a formula φ_m in $\mathsf{MSO}(\mathfrak{A}, \Sigma)$ characterising the under-approximation then the automaton $\mathcal{A}^k_{\varphi_m}$ serves as $\mathcal{A}^k_{\mathcal{C}_m}$.

[4] If $\mathcal{C}_m$ is recognised by a CPDS $\mathcal{S}_m$ of size $2^{\mathsf{poly}(k)}$, then the automaton $\mathcal{A}^k_{\mathcal{C}_m} = \mathcal{A}^k_{\mathcal{S}_m}$ is of size $2^{\mathsf{poly}(k)}$.

problems but also for a wide range of classes. The complexities stated in Table 2 match the lower-bounds for many known under-approximation classes, thus asserting the optimality of the uniform decision procedures. For details we refer to [15, Section 4.4].

Word-like. A split-decomposition is said to be word-like if for every binary node in the decomposition tree, one of its subtrees has depth bounded by a constant. In this case, we could employ word automata instead of tree automata. All behaviours of some under-approximation classes, like existentially m-bounded, admit a word-like split decomposition. For many problems, the complexity upper bounds fall to the maximal space-complexity classes contained in the time-complexity classes. For example, emptiness checking of a CPDS parametrised by (word-like) split-width k can be done in PSpace instead of ExpTime, and if k is fixed, in NLogSpace instead of PTime.

Acknowledgement. The authors thank Benedikt Bollig for many fruitful discussions and constructive comments.

References

1 C. Aiswarya, P. Gastin, and K. Narayan Kumar. Controllers for the verification of communicating multi-pushdown systems. In *CONCUR'14*, volume 8704 of *LNCS*, pages 297–311. Springer, 2014.

2 C. Aiswarya, P. Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *ATVA'14*, volume 8837 of *LNCS*. Springer, 2014. To appear.

3 R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. *Log. Meth. Comput. Sci.*, 4(4:11), 2008.

4 R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.

5 R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3:16), 2009.

6 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-Complete. In *DLT'08*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.

7 B. Bollig, A. Cyriac, P. Gastin, and M. Zeitoun. Temporal logics for concurrent recursive programs: Satisfiability and model checking. *Journal of Applied Logic*, 2014. To appear.

8 B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3:16), 2010.

9 B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, 358(2):150–172, 2006.

10 L. Breveglieri, A. Cherubini, Cl. Citrini, and S. Crespi-Reghizzi. Multi-pushdown languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

11 J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

12 P. Chambart and Ph. Schnoebelen. Mixing lossy and perfect FIFO channels. In *CONCUR'08*, volume 5201 of *LNCS*, pages 340–355. Springer, 2008.

13 L. Clemente, F. Herbreteau, and G. Sutre. Decidable topologies for communicating automata with FIFO and bag channels. In *CONCUR'14*, volume 8704 of *LNCS*, pages 281–296. Springer, 2014.

14 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

15 A. Cyriac. *Verification of Communicating Recursive Programs via Split-width.* PhD thesis, ENS Cachan, 2014. `http://www.lsv.ens-cachan.fr/~cyriac/download/Thesis_Aiswarya_Cyriac.pdf`.

16 A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR'12*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.

17 C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

18 B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.

19 A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FOSSACS'10*, volume 6014 of *LNCS*, pages 267–281. Springer, 2010.

20 ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, February 2011.

21 S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE Computer Society Press, 2007.

22 S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS'08*, volume 4963 of *LNCS*, pages 299–314. Springer, 2008.

23 S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR'11*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

24 S. La Torre and M. Napoli. A temporal logic for multi-threaded programs. In *IFIP TCS*, volume 7604 of *LNCS*, pages 225–239. Springer, 2012.

25 S. La Torre, M. Napoli, and G. Parlato. A unifying approach for multistack pushdown automata. In *MFCS'14*, volume 8634 of *LNCS*, pages 377–389. Springer, 2014.

26 P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 283–294. ACM, 2011.

27 A. Muscholl. *Über die Erkennbarkeit unendlicher Spuren.* Teubner, 1996.

28 D. Peled and Th. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63(5):243–246, 1997.

29 D. Peled, Th. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and ω-regular languages. *Theoretical Computer Science*, 195(2):183–203, 1998.

30 Sh. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

31 J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

32 B. A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 149:326–329, 1961.

Colour Refinement: A Simple Partitioning Algorithm with Applications From Graph Isomorphism Testing to Machine Learning

Martin Grohe

RWTH Aachen University
Aachen, Germany
`grohe@informatik.rwth-aachen.de`

Abstract

Colour refinement is a simple algorithm that partitions the vertices of a graph according their "iterated degree sequence." It has very efficient implementations, running in quasilinear time, and a surprisingly wide range of applications. The algorithm has been designed in the context of graph isomorphism testing, and it is used an important subroutine in almost all practical graph isomorphism tools. Somewhat surprisingly, other applications in machine learning, probabilistic inference, and linear programming have surfaced recently.

In the first part of my talk, I will introduce the basic algorithm as well as higher dimensional extensions known as the k-dimensional Weisfeiler-Lehman algorithm. I will also discuss an unexpected connection between colour refinement and a natural linear programming approach to graph isomorphism testing. In the second part of my talk, I will discuss various applications of colour refinement.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Color Refinement, Graph Isomorphism, Machine Learning

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.31

Category Invited Talk

Properties and Utilization of Capacitated Automata

Orna Kupferman[1] **and Tami Tamir**[2]

1 **School of Engineering and Computer Science, Hebrew University, Israel**
orna@cs.huji.ac.il

2 **School of Computer Science, The Interdisciplinary Center, Herzliya, Israel**
tami@idc.ac.il

Abstract

We study *capacitated automata* (CAs), where transitions correspond to resources and may have bounded capacities. Each transition in a CA is associated with a (possibly infinite) bound on the number of times it may be traversed. We study CAs from two points of view. The first is that of traditional automata theory, where we view CAs as recognizers of formal languages and examine their expressive power, succinctness, and determinization. The second is that of resource-allocation theory, where we view CAs as a rich description of a flow network and study their utilization.

1998 ACM Subject Classification F.4.3 Formal Languages, B.8.2 Performance Analysis and Design Aids, F.1.1 Models of Computation

Keywords and phrases Automata, Capacitated transitions, Determinization, Maximum utilization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.33

Category Invited Talk

1 Introduction

Finite state automata are used in the modeling and design of finite-state systems and their behaviors, with applications in engineering, databases, linguistics, biology, and many more. The traditional definition of an automaton does not refer to its transitions as consumable resources. Indeed, a run of an automaton is a sequence of successive transitions, and there is no bound whatsoever on the number of times that a transition may be traversed. In practice, the use of a transition may correspond to the use of some resource. For example, it may be associated with the usage of some energy-consuming machine, application of some material, or consumption of bandwidth. We study *capacitated automata* (CAs). In this model, transitions correspond to resources and may have bounded capacities. Formally, each transition is associated with a (possibly infinite) integral bound on the number of times it may be traversed. A word w is accepted by a nondeterministic capacitated automaton (NCA) $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on w; one that reaches an accepting state and respects the bounds on the transitions.

We examine CAs from two points of view. The first, more related to traditional automata theory, views CAs as recognizers of formal languages. The interesting questions that arise in this view are similar to classical questions about automata: their expressive power, succinctness, determinization, decision problems, etc. The second view, more related to traditional resource-allocation theory, views CAs as labeled flow networks. The interesting questions then have to do with optimal utilization of the system modeled by the CA.

Let us start with the first view. In terms of expressive power, we show that capacities can be removed. Thus, NCAs are not more expressive than nondeterministic finite automata (NFAs), and can recognize exactly all regular languages. The main questions that arise, then, refer to the succinctness of the capacitated model with respect to the standard one, as well as the blow-up involved in their determinization. Consider, for example, the language $L_{n,m}$ over the alphabet $\Sigma_n = \{1, \ldots, n\}$ that contains exactly all words in which each letter in Σ_n appears at most m times. It is not hard to see that a traditional, possibly nondeterministic, automaton for $L_{n,m}$ needs at least m^n states. On the other hand, a deterministic capacitated automaton (DCA) for L_n consists of a single state with n self-loops, each labeled by a different letter from Σ_n and has capacity m. Hence, both NCAs and DCAs may be exponentially more succinct than NFAs. The nondeterministic model, however, is much stronger, and determinization involves a blow-up that is not only exponential in the state space but also linear in the product of the capacities. Note that the latter is at least exponential in the number of transitions. This is surprising, as we do allow the obtained DCA to have capacities. As we show, there are languages, in particular strongly liveness languages [1], for which the power of capacities is significant in the nondeterministic model but is not realized in the deterministic one.

We then turn to solve decision problems about CAs. Two classical problems are the *nonemptiness* (does $\mathcal{A}$ accept at least one word?) and the *membership* (does $\mathcal{A}$ accept a given word w?) problem. In the traditional model, the problems are essentially the same: checking whether $\mathcal{A}$ accepts w can be reduced to checking the emptiness of the product of $\mathcal{A}$ with w. Accordingly, both problems can be reduced to reachability. In the capacitated model, taking the product of a word with an automaton may require tracing the history of traversals, and indeed we prove that while the nonemptiness problem is not more difficult than in the traditional model, membership becomes NP-complete for NCAs. Moreover, if we augment the nonemptiness problem to consider words from a given language (that is, given a CA $\mathcal{A}$ and a regular language L, decide whether $\mathcal{A}$ accepts some word from L), it becomes NP-complete already for DCAs, even when L is given by a DFA.

We continue to the second view, where CAs model labeled flow networks. In order to motivate this view, let us first demonstrate the different ways in which we consider CAs in the two views. Consider the containment problem for CAs, asking whether a given set S of words is contained in the language of a CA $\mathcal{A}$. We can think of two variants of the problem. In the first, which corresponds to our first view, we ask whether $\mathcal{A}$ accepts w for each word $w \in S$. In the second, which corresponds to our second view, we ask whether $\mathcal{A}$ *mutually* accepts all words in S. That is, whether $\mathcal{A}$ has enough capacity to process all the words in S mutually. It is not surprising that a CA may contain S in the first view but not in the second. Consider for example the NCA $\mathcal{A}$ described in Figure 1. Let $S = \{ab, abc, ac, abb, abcb\}$. Clearly, all the words in S are accepted by $\mathcal{A}$, thus $\mathcal{A}$ contains S in the first view. On the other hand, there is no way for $\mathcal{A}$ to mutually accept all the words in S. Indeed, it is not hard to see that $\mathcal{A}$ can make only a single use of the edge $\langle q_1, c, q_2 \rangle$ even though its capacity is 2, whereas S contains three words with the letter c. Note that $\mathcal{A}$ can mutually accept the set $S' = \{ab, ac, abb, abcb\}$, but one has to carefully resolve nondeterminism in order to do it. For example, once $\mathcal{A}$ processes abb via q_1, it can no longer process both ac and $abcb$. Thus, in the second view, the challenge is to find ways to mutually accept in a given NCA as many words as possible, as we formally define below.

A natural problem that arises when reasoning about systems that consist of resources

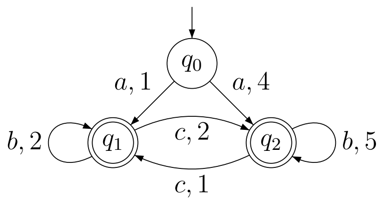


Figure 1 An NCA that contains $S = \{ab, abc, ac, abb, abcb\}$ in the traditional view but does not contain S mutually.

with limited capacities is to utilize these resources in the best way. In our model, the *max-utilization problem* is defined as follows. Given a CA $\mathcal{A}$, return a multiset W of words, such that $\mathcal{A}$ mutually accepts all the words in W, and $|W|$ is maximal. The max-utilization problem can be viewed as a generalization of the max-flow problem in networks [11]. In the max-flow problem, the network is utilized by units of flow, each routed from the source to the target. The CA model enables a rich description of the feasible routes. The labels along a path correspond to a sequence of applications of resources. In particular, paths from an initial state to a final state correspond to feasible such sequences, and the goal is to mutually process as many of them as possible.

Sometimes, not all the sequences feasible in the NCA are desirable. Accordingly, we also consider the *max restricted-utilization problem*, where the input to the problem also includes a language specifying the desirable sequences. Then, the words in the multiset W must belong to this language. For example, the language may restrict the length of the sequences, preventing long sequences from consuming the system (see [17] for an analogous restriction in flow networks), it may restrict the number of different resources applied in a sequence, it may require an application of specific resources, it may require a specific event to trigger another specific event, and so on. Thus, while in traditional flow problems the specification of desired routes is given by means of a source and a target, the max restricted utilization problem enables specifications that are much richer than reachability. A similar lifting of reachability was studied in [3], where network formation games were extended to automata formation games. We study the complexity of the max utilization problems and show that while the unrestricted variant can be solved in polynomial time by a simple reduction to a network maximum-flow problem, even simple restrictions on the desirable routes make the problem hard to approximate. Essentially, this follows from the fact that the basic idea of an augmenting path in a network cannot be adopted to NCAs. Indeed, in NCAs every path corresponds to a sequence of applications of recourses, and the augmenting path need not correspond to a desired such sequence.

Related Work

Many extensions of automata in which transitions are augmented by numerical values have been studied. Most notable are *probabilistic automata* [19], where the values form a distribution on the successor state, and *weighted automata* [10], where weights are used in order to model costs, rewards, certainty, and many more. The semantics of these models is multi-valued: each traversal of a transition updates some accumulated value, and the language of the automaton maps words into some domain. In particular, the B and S automata of [8] count traversals on transitions. Our semantics, on the other hand, maintains the Boolean nature of regular languages, and only augments the way in which acceptance is defined.

More related to our work are extensions of automata that stay in the Boolean domain, in particular ones whose semantics involves counting. *Parikh automata* were introduced and studied by Klaedtke and Rueß in [16]. Their semantics involves counting of the number of occurrences of each letter in Σ in the word. Essentially, a Parikh automaton is a pair $\langle \mathcal{A}, C \rangle$, where $\mathcal{A}$ is an NFA over Σ, and $C \subseteq \mathbb{N}^{\Sigma}$ is a set of "allowed occurrences". A word w is accepted by $\langle \mathcal{A}, C \rangle$ if both $\mathcal{A}$ accepts w and the Parakh's commutative image of w, which maps each letter in Σ to its number of occurrences in w, is in C. It is easy to see that the expressive power of Parikh automata goes beyond regular language. For example, Parikh automata can recognize the language $\{a^i b^i : i \geq 1\}$ by defining $\mathcal{A}$ to recognize $a^* b^*$ and defining C to contain all pairs $\langle i, i \rangle$, for $i \geq 1$. In fact, by [15], Parikh automata are as expressive as reversal-bounded counter machines [13].

Several variants of Parikh automata have seen studied. In particular, [7] studied *constrained automata*, a variant that counts traversals of transitions and requires the vector of counters to belong to C, now a semilinear set of allowed vectors. NCAs can be viewed as a special case of constrained automata in which C is downwards closed. This significantly restricts the expressive power of constrained automata, and indeed the types of questions we consider are different than these studied for Parikh automata and their variants.

Additional strictly more expressive models include *multiple counters automata* [9], where transitions can be taken only if guards referring to traversals so far are satisfied, and *queue-content decision diagrams*, which are used to represent queue content of FIFO-channel systems [5, 6]. Finally, a model with the same name – *finite capacity automata* is used in [18] in order to model the control of an automated manufacturing system. This model is different from our CAs and is more related to Petri nets.

The above works take the first view on automata, namely study them as recognizers of languages. As for the second view, a lot of research has been done on optimal utilization of limited resources. In particular, as discussed above, max-utilization of a CA corresponds to a maximum flow in a network [11]. By restricting the domain from which accepted words can be chosen, we get an intractable problem. This resemble the intractability of some variants of the max-flow problem, such as k-bounded flow [4], or max-flow with bounded-length paths [17].

2 Preliminaries

A *nondeterministic finite automaton* (NFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is a set of final states. Given a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_l$, a *run* of $\mathcal{A}$ on w is a sequence r of successive transitions in Δ that reads w and starts in a transition from the set of initial states. Thus, $r = \langle q_0, \sigma_1, q_1 \rangle, \langle q_1, \sigma_2, q_2 \rangle, \ldots, \langle q_{l-1}, \sigma_l, q_l \rangle$, for $q_0 \in Q_0$. The run is accepting if $q_l \in F$. We sometimes refer to the transitions function $\delta : Q \times \Sigma \to Q$ induced by Δ, thus $q' \in \delta(q, \sigma)$ iff $\Delta(q, \sigma, q')$. The NFA $\mathcal{A}$ *accepts* the word w iff it has an accepting run on it. Otherwise, $\mathcal{A}$ *rejects* w. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$ is the set of words that $\mathcal{A}$ accepts. If $|Q_0| = 1$ and for all $q \in Q$ and $\sigma \in \Sigma$ there is at most one $q' \in Q$ with $\Delta(q, \sigma, q')$, then $\mathcal{A}$ is *deterministic*. Note that a deterministic finite automaton (DFA) has at most one run on each word.

A *nondeterministic capacitated automaton* (NCA, for short) is an NFA in which each transition has a *capacity*, bounding the number of times it may be traversed. A transition may not be bounded, in which case its capacity is ∞. Let $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. Formally, an NCA is a pair $\langle \mathcal{A}, c \rangle$, where $\mathcal{A}$ is an NFA and $c : \Delta \to \mathbb{N}^\infty$ is a capacity function that maps each transition in Δ to its capacity. A run of $\langle \mathcal{A}, c \rangle$ is a run of $\mathcal{A}$ in which the number of occurrences of each transition $e \in \Delta$ is at most $c(e)$. When $\mathcal{A}$ is deterministic, then so is $\langle \mathcal{A}, c \rangle$. For a CA $\mathcal{A}$ and a set of words S, we say that $\mathcal{A}$ *mutually accepts* S if there are $|S|$ accepting runs, one for each word in S, such that for each transition $e \in \Delta$, the total number of occurrences of e in all these runs is at most $c(e)$.

For a capacity function $c : \Delta \to \mathbb{N}^\infty$, let $c_\downarrow$ be the set of capacity functions obtained by closing c downwards. Formally, a function $c' : \Delta \to \mathbb{N}^\infty$ is in $c_\downarrow$ if for all transitions $e \in \Delta$ with $c(e) = \infty$, we have $c'(e) = \infty$, and for all transitions $e \in \Delta$ with $c(e) \in \mathbb{N}$, we have $0 \le c'(e) \le c(e)$. It is easy to see that the size of $c_\downarrow$, denoted $|c_\downarrow|$, is $\Pi_{e:c(e)\in\mathbb{N}^+}(c(e)+1)$. Thus, $|c_\downarrow|$ is exponential in the number of transitions with bounded capacities.

3 Theoretical Properties of NCAs

In this section we study the expressive power and succinctness of NCAs with respect to NFAs, as well as their determinization.

3.1 Capacities Removal

▶ **Theorem 1.** *NCAs accept regular languages: Every NCA $\langle \mathcal{A}, c \rangle$ has an equivalent NFA $\mathcal{A}'$. The size of $\mathcal{A}'$ is linear in the size of $\mathcal{A}$ and $|c_\downarrow|$.*

Proof. Given an NCA $\langle \mathcal{A}, c \rangle$ with $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F \rangle$, we define an equivalent NFA $\mathcal{A}' = \langle \Sigma, Q', Q'_0, \Delta', F' \rangle$ as follows.

- $Q' = Q \times c_\downarrow$ and $Q'_0 = Q_0 \times \{c_\downarrow\}$. That is, each state in $\mathcal{A}'$ maintains both the corresponding state in $\mathcal{A}$ and the capacities that are left to be consumed.
- For $q, q' \in Q$, $d, d' \in c_\downarrow$, and $\sigma \in \Sigma$, we have that $\Delta'(\langle q, d \rangle, \sigma, \langle q', d' \rangle)$ iff $\Delta(q, \sigma, q')$, $d(\langle q, \sigma, q' \rangle) > 0$, $d'(e) = d(e)$ for all $e \neq \langle q, \sigma, q' \rangle$, and $d'(\langle q, \sigma, q' \rangle) = d(\langle q, \sigma, q' \rangle) - 1$. That is, d' is updated to take the traversal of $\langle q, \sigma, q' \rangle$ into an account by reducing its capacity by 1.
- $F' = F \times c_\downarrow$.

Note that the construction preserves determinism, thus if $\mathcal{A}$ is a DCA, the obtained $\mathcal{A}'$ is a DFA. It is not hard to prove that each run r of $\mathcal{A}$ corresponds to a run of $\mathcal{A}'$, obtained by pairing each state in r by the capacity function that reflects the updates to c according to the transitions traversed so far. Dually, each run r' of $\mathcal{A}'$ corresponds to a run of $\mathcal{A}$, obtained by projecting r' on the Q-elements of its states. By the definition of $\mathcal{A}'$, the obtained run respects the bounds on the transitions.

We prove that the blow-up in $|c_\downarrow|$ cannot be avoided. We prove it already for single-state DCAs. Given two parameters $n, m \in \mathbb{N}$, consider the DCA $\langle \mathcal{A}_{n,m}, c_{n,m} \rangle$, where $\mathcal{A}_{n,m} = \langle \{1, \ldots, n\}, \{q\}, \{q\}, \Delta, \{q\} \rangle$ is such that each letter $i \in \{1, \ldots, n\}$ contributes to Δ the transition $\langle q, i, q \rangle$. That is, $\mathcal{A}_{n,m}$ consists of a single state with one self-loop transition for each of the n letters. The capacity of all transitions is m. Thus, $c_{n,m}(\langle q, i, q \rangle) = m$ for all $i \in \{1, \ldots, n\}$. It is easy to see that the language $L_{n,m}$ of $\langle \mathcal{A}_{n,m}, c_{n,m} \rangle$ is the set of all words in which each of the letters $\{1, \ldots, n\}$ appears at most m times, and that every NFA that recognizes $L_{n,m}$ needs at least n^m states. ◀

Theorem 1 implies that, like regular languages, NCAs and DCAs are closed under union, intersection, and complementation, and that NCAs can be determinized. The question is

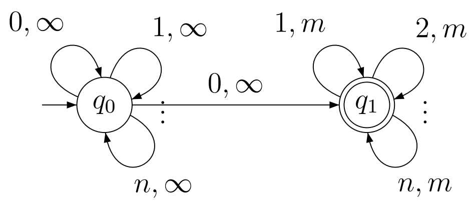


Figure 2 An NCA for $L'_{n,m}$.

the blow-up involved in the corresponding constructions, in particular whether they need to involve removal of capacities.

3.2 Determinization

In this section we study the succinctness of NCAs with respect to DCAs. We show that NCAs are exponentially more succinct not only in the number of states but also in the number of transitions. More precisely, determination may involve a blow-up linear in $c_\downarrow$. This is surprising, as we do allow the obtained deterministic automaton to be capacitated. We first prove that there are languages for which capacities are not useful in the deterministic setting. A language $L \subseteq \Sigma^*$ is *strongly liveness* if $L = \Sigma^* \cdot L$ [1]. Thus, in terms of temporal logic, strongly liveness languages correspond to properties if the form "eventually ψ" for some behavior ψ.

▶ **Lemma 2.** *A DCA for a strongly liveness language L is not smaller than a DFA for L.*

Proof. Consider a DCA $\langle \mathcal{A}, c\rangle$ that recognizes L. We say that a word $w \in \Sigma^*$ *consumes* $\langle \mathcal{A}, c\rangle$ if, after reading w, a run of $\mathcal{A}$ can proceed only along transitions with infinite capacity (or cannot proceed at all). It is easy to see that there exists at least one word w that consumes $\langle \mathcal{A}, c\rangle$. Let q be the state that $\mathcal{A}$ reaches by reading w. Since $L = \Sigma^* \cdot L$, we have that $w \cdot L \subseteq L$. Hence, since $\mathcal{A}$ is deterministic, the language of the DFA $\mathcal{A}'$ obtained from $\mathcal{A}$ by making q its initial state and by removing all transitions that do not have infinite capacity is L. The size of $\mathcal{A}'$ is at most the size of $\mathcal{A}$, and we are done. ◀

▶ **Theorem 3.** *Determinization of NCAs involves a blow-up exponential in Q and linear in $c_\downarrow$.*

Proof. The exponential blow-up with respect to Q follows from the known exponential blow-up in determinization of NFAs [20]. In order to prove the blow-up in $c_\downarrow$, we describe a family $L'_{n,m}$, for $n, m \geq 1$ of strongly liveness languages such that $L'_{n,m}$ is over the alphabet $\Sigma_n = \{0, \ldots, n\}$, it can be recognized by a two-state NCA with n transitions with capacity m. By Lemma 2, the size of a DCA for $L'_{n,m}$ is equal to the size of a DFA for it, which is at least m^n.

Let $L_{n,m}$ be the language of all words w over $\{1, \ldots, n\}$ such that each letter in $\{1, \ldots, n\}$ appears in w at most m times (the same language as in the proof of Theorem 1). We define $L'_{n,m} = \Sigma_n^* \cdot 0 \cdot L_{n,m}$, Clearly, $L'_{n,m}$ is a strongly liveness language, and it can be recognized by the two-state NCA described in Figure 2. ◀

4 Decision Problems

In this section we study the following decision problems for NCAs and DCAs. Below we also state their known complexity in the traditional setting (see, for example [12]).

- The *nonemptiness* problem: given an automaton $\mathcal{A}$, decides whether $L(\mathcal{A}) \neq \emptyset$. For both NFA and DFA, the nonemptiness problem is NLOGSPACE-complete.
- The *membership* problem: given an automaton $\mathcal{A}$ and a finite word w, decide whether $w \in L(\mathcal{A})$. For NFA and DFA, the membership problem is NLOGSPACE-complete and LOGSPACE-complete, respectively.
- The *relative nonemptiness* problem: given an automaton $\mathcal{A}$ and a language L, decide whether $\mathcal{A}$ accepts at least one word from L. The relative nonemptiness problem for an NFA or a DFA $\mathcal{A}$ and a language L given by an NFA or a DFA is NLOGSPACE-complete.

▶ **Theorem 4.** *The nonemptiness problem for NCAs and DCAs is NLOGSPACE-complete.*

Proof. An NCA is nonempty iff there is a simple path from some initial state to some accepting state. Since the path is simple, capacities do not play a role (beyond exclusion of transitions with capacity 0). Thus, nonemptiness can be reduced to reachability, implying membership in NLOGSPACE. The lower bound follows from NLOGSPACE hardness for DFA emptiness. ◀

▶ **Theorem 5.** *The membership problem can be solved in linear time for DCAs and is NP-complete for NCAs.*

Proof. We start with the upper bounds. Given a DCA $\langle \mathcal{A}, c\rangle$ and a word w, we can trace the single run of $\langle \mathcal{A}, c\rangle$ on w and check that it ends in an accepting state and respects c. When $\langle \mathcal{A}, c\rangle$ is an NCA, a witness to the membership of w in $\langle \mathcal{A}, c\rangle$ is an accepting run of $\langle \mathcal{A}, c\rangle$ on w. As above, it can be checked in linear time.

For NCAs, we prove hardness in NP already for the case $|\Sigma| = 1$. We describe a reduction from the problem of deciding whether a given directed graph has a Hamiltonian cycle – one that visits all vertices of the graph exactly once. Given a graph $G = \langle V, E\rangle$, we construct the NCA $\langle \mathcal{A}, c\rangle$ as follows (see an example in Figure 3. The graph G is on the left, the NCA $\mathcal{A}$ is in the middle). Let v_1 be some vertex in V. Then, $\mathcal{A} = \langle \{\sigma\}, V \times \{\text{in,out}\}, \{\langle v_1, \text{out}\rangle\}, \Delta, \{\langle v_1, \text{out}\rangle\}\rangle$ is such that each vertex $v \in V$ contributes to Δ the transition $\langle\langle v, \text{in}\rangle, \sigma, \langle v, \text{out}\rangle\rangle$ with capacity 1, and each edge $\langle u, v\rangle$ in E contributes to Δ the transition $\langle\langle u, \text{out}\rangle, \sigma, \langle v, \text{in}\rangle\rangle$, again with capacity 1. It is not hard to see that G has a Hamiltonian cycle iff $\langle \mathcal{A}, c\rangle$ accepts the word $\sigma^{2|V|}$. Indeed, having capacity 1 on the transitions that correspond to vertices in V guarantees that each vertex is visited at most once, thus a word of length $2|V|$ must close a cycle and visits exactly all vertices in G. ◀

▶ **Theorem 6.** *The relative nonemptiness problem for NCAs and DCAs relative to languages given by NCAs, DCAs, NFAs, or DFAs is NP-complete.*

Proof. Consider a capacitated automaton $\langle \mathcal{A}, c\rangle$ and an automaton $\mathcal{U}$ such that we want to check the nonemptiness of $\mathcal{A}$ with respect to $L(\mathcal{U})$. Note that we have eight cases to consider, reflecting whether $\mathcal{A}$ is an NCA or a DCA and whether $\mathcal{U}$ is an NCA, DCA, NFA, or DFA. We prove that all eight cases are NP-complete.

We start with membership in NP for the most general case, where both $\langle \mathcal{A}, c\rangle$ and $\mathcal{U} = \langle \mathcal{A}', c'\rangle$ are NCAs. A witness to the relative nonemptiness is a word w and accepting runs of $\langle \mathcal{A}, c\rangle$ and $\langle \mathcal{A}', c'\rangle$ on it. The word w does not traverse cycles in the product of $\mathcal{A}$

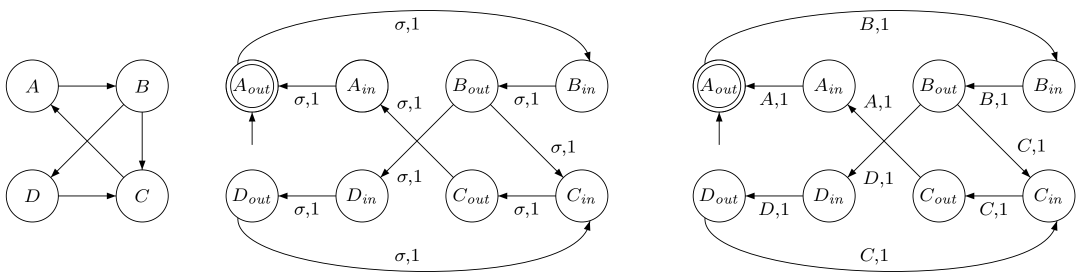


Figure 3 Two reductions from the Hamiltonian cycle problem.

and $\mathcal{A}'$. It is thus of polynomial length in the product, which is of size $|\mathcal{A}| \cdot |\mathcal{A}'|$. Checking that the runs respect c and c' can also be done in polynomial time.

We prove hardness in NP for the most restricted case, where $\langle \mathcal{A}, c\rangle$ is a DCA and $\mathcal{U}$ is a DFA. We describe a reduction from the problem of deciding whether a given directed graph has a Hamiltonian cycle (see an example in Figure 3. The graph G is on the left, the NCA $\mathcal{A}$ is on the right). Given $G = \langle V, E\rangle$ with $V = \{v_1, \ldots, v_n\}$, we construct $\mathcal{A} = \langle V, V \times \{\text{in,out}\}, \{\langle v_1, \text{out}\rangle\}, \Delta, \{\langle v_1, \text{out}\rangle\}\rangle$, where v_1 is an arbitrary vertex in V. Each vertex $v \in V$ contributes to Δ the transition $\langle\langle v, \text{in}\rangle, v, \langle v, \text{out}\rangle\rangle$ with capacity 1, and each edge $\langle u, v\rangle$ in E contributes to Δ the transition $\langle\langle u, \text{out}\rangle, v, \langle v, \text{in}\rangle\rangle$ with capacity 1. It is easy to see that $\mathcal{A}$ is indeed a DCA and that G has a Hamiltonian cycle iff $\mathcal{A}$ accepts a word in the language $[(v_1 \cdot v_1) + \cdots + (v_n \cdot v_n)]^n$, which can be recognized by a DFA with $O(n^2)$ states. In the example, the Hamiltonian cycle $ABDCA$ corresponds to the word $BBDDCCAA$. ◀

5 The Maximum Utilization Problem

A natural problem that arises when reasoning about systems that consist of resources with limited capacities is to utilize these resources in the best way. In our model, the maximum utilization problem is defined as follows. Given a CA $\langle \mathcal{A}, c\rangle$, return a multiset W of words, such that $\langle \mathcal{A}, c\rangle$ has enough capacity to mutually accept all the words in W, and $|W|$ is maximal. We refer to W as an *optimal utilization mutiset* for $\langle \mathcal{A}, c\rangle$.

As discussed in Section 1, the max-utilization problem can be viewed as a generalization of the max-flow problem in a network. The CA model enables a rich description of the feasible routes. Sometimes, not all the sequences allowed by the CA are desirable. Accordingly, we also consider the *max restricted utilization problem*, where the input to the problem also includes a language L, specifying the desirable sequences. Then, the words in the optimal utilization multiset must belong to L. In Section 5.1, we show that the unrestricted max-utilization problem can be solved in polynomial time by a reduction to the classical network-flow problem [11]. In section 5.2 we study the restricted case and show that adding restrictions makes the problem much more complex.

5.1 Maximum Unrestricted Utilization

We present an optimal algorithm for the max-utilization problem. The algorithm is based on a reduction to a max-flow problem in a network. Recall that a max-flow problem is defined over a flow-network given by a directed graph $G = \langle V, E\rangle$, two vertices $s, t \in V$ designated as the source and the target vertices, and a capacity function c that maps each edge $e \in E$ to a positive integral capacity $c(e)$. For a vertex v, let $in(v)$ and $out(v)$ denote the set of edges into and out of v, respectively. A *legal flow* is a function $f : E \to \mathbb{R}$ such that for every edge $e \in E$, it holds that $0 \leq f(e) \leq c(e)$, and for every vertex $v \in V \setminus \{s,t\}$, it holds that $\sum_{e\in in(v)} f(e) = \sum_{e\in out(v)} f(e)$. A max-flow is a legal flow that maximizes the flow leaving the source, given by $\sum_{e\in out(s)} f(e) - \sum_{e\in in(s)} f(e)$.

▶ **Theorem 7.** *The max-utilization problem for NCAs and DCAs can be solved in polynomial time.*

Proof. Let $\langle \mathcal{A}, c\rangle$ be an NCA, with $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F\rangle$. If $\mathcal{A}$ has a path from Q_0 to F all whose transitions have infinite capacity (in particular, if $Q_0 \cap F \neq \emptyset$), then, as the optimal utilization multiset is not restricted, so is the max-flow, and we return as a witness the (possibly empty) word along the above accepting run.

Otherwise, the optimal utilization multiset must be finite and we proceed as follows. Given $\langle \mathcal{A}, c\rangle$, we construct a flow-network $G = \langle Q \cup \{s,t\}, E\rangle$, where every transition $\langle p, \sigma, q\rangle \in \Delta$ induces an edge $\langle p, q\rangle$ in E with capacity $c(\langle p, \sigma, q\rangle)$. The set E includes also the set of edges $\{s\} \times Q_0$ and $F \times \{t\}$, all with an unbounded capacity. It is easy to see that a max-flow in G corresponds to a maximal multiset of words that can be mutually processed by $\langle \mathcal{A}, c\rangle$. The optimal utilization multiset W can be obtained by tracking the (s,t)-paths in the max-flow. Note that since there are no restriction on the words in W, the alphabet in $\langle \mathcal{A}, c\rangle$ plays no role. ◀

5.2 Maximum Restricted Utilization

In the max restricted-utilization problem, we are given, in addition to a CA $\langle \mathcal{A}, c\rangle$, also a regular language L. The optimal utilization multi set W then has to contain only words from L. We refer to L as the *restricting language.*

We present a polynomial-time optimal algorithm for the max restricted-utilization problem for the case the restricting language consists of words of length at most 2. This is indeed a very limited class of languages. Yet, it is tight, as we provide an APX-hardness proof already for the case that $\langle \mathcal{A}, c\rangle$ is a DCA with $c \equiv 1$ and the restricting language consists of words of length 3.

▶ **Theorem 8.** *When all the words in the restricting language are of length at most* 2*, the max restricted-utilization problem can be solved in polynomial time.*

Proof. Let $\langle \mathcal{A}, c\rangle$ be an NCA and L a restricting language all of whose words are of length at most 2. First, if $\varepsilon \in L(\langle \mathcal{A}, c\rangle) \cap L$, then the optimal utilization multiset is infinite, and we are done. Hence, we assume that all words in L are of length 1 or 2. We present an optimal algorithm that is based on reducing the problem to a *maximum b-matching* problem. An instance of b-matching consists of an undirected graph $G = \langle V, E\rangle$ and a budget function $b : V \to \mathbb{N}$. A b-matching is an assignment of a non-negative integer weight x_e to every edge $e \in E$, such that for every vertex $v \in V$, the sum of weights on edges incident with v is at most $b(v)$. The maximum b-matching problem is to find a matching of maximum profit; that is, $\max \sum_e x_e$. The graph G need not be simple. That is, self-loops and parallel edges are allowed. By [2], the maximum b-matching problem can be solved in time polynomial in $|V|$ and $|E|$.

Let $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F\rangle$, and let $W_1(L)$ and $W_2(L)$ denote the words of length 1 and 2 in L, respectively. For every $\sigma \in W_1(L)$, let $A_\sigma \subseteq \Delta$ be the set of transitions that form an accepting run on σ in $\langle \mathcal{A}, c\rangle$. Formally, $e \in A_\sigma$ iff $e = \langle q_0, \sigma, p\rangle$ for $q_0 \in Q_0$, $p \in F$, and $c(e) \geq 1$. For every $w = \sigma_1 \cdot \sigma_2 \in W_2(L)$, let $A_w \subseteq \Delta \times \Delta$ be the set of pairs of transitions that

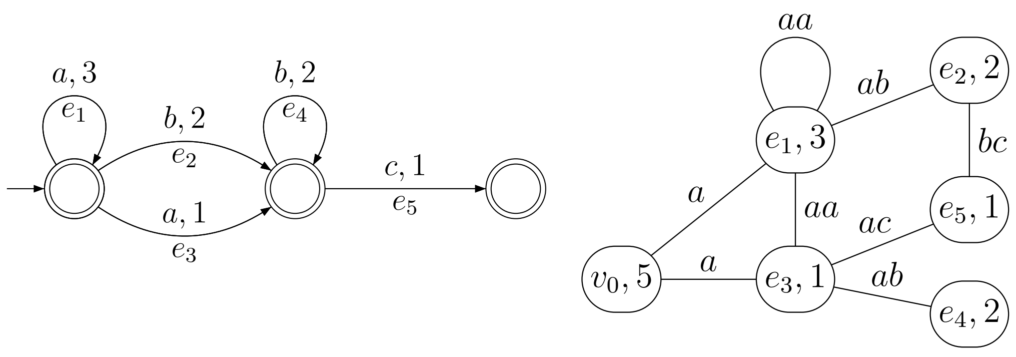


Figure 4 An NCA and the b-matching instance constructed for $L = \{a, aa, ab, ac, ba, bc, cc\}$.

form an accepting run on w in $\mathcal{A}$. Formally, $\langle e_1, e_2\rangle \in A_w$ iff $e_1 = \langle q_1, \sigma_1, p_1\rangle, e_2 = \langle q_2, \sigma_2, p_2\rangle$ for $q_1 \in Q_0$, $p_1 = q_2$, $p_2 \in F$, $c(e_1) \geq 1$, and $c(e_2) \geq 1$. If $e_1 = e_2$, then we also require $c(e_1) \geq 2$. Note that if $w \notin L(\mathcal{A})$, then E_w is empty. Also, if w has several accepting runs in $\mathcal{A}$, then $|E_w| > 1$. It is possible to compute the above sets in time $O(|Q|^2)$.

We construct an undirected graph $G = \langle \Delta \cup \{v_0\}, E\rangle$, where E consists of edges of two types, defined as follows.

1. $\langle e_i, v_0\rangle \in E$ iff there exists a word $\sigma \in W_1(L)$ such that $e_i \in A_\sigma$.
2. $\langle e_i, e_j\rangle \in E$ iff there exists a word $w \in W_2(L)$ such that $\langle e_i, e_j\rangle \in A_w$.

Every edge of type (e_i, v_0) corresponds to a 1-letter word. Every edge of type (e_i, e_j) corresponds to at least one 2-letter word. Since Q_0 may include several initial states, it is possible that both $\langle e_i, e_j\rangle$ and $\langle e_j, e_i\rangle$ form accepting runs in $\langle \mathcal{A}, c\rangle$. It is also possible to have a self-loop in G if $\mathcal{A}$ includes a self-loop from the initial state.

An example of the reduction is given in Figure 4. The CA $\langle \mathcal{A}, c\rangle$ is on the left and the b-matching instance constructed for $L = \{a, aa, ab, ac, ba, bc, cc\}$ is on the right. Note that each of the words a, aa, and ab has two possible accepting runs in $\langle \mathcal{A}, c\rangle$. Thus, each of these words induces two edges in G. Words that belong to $L \setminus L(\langle \mathcal{A}, c\rangle)$, like ba and cc, as well as words that belong to $L(\langle \mathcal{A}, c\rangle) \setminus L$, like b or bb, do not induce an edge in G.

To complete the definition of the b-matching instance, we set the vertex budgets as follows. For every $e_i \in \Delta$, we set $b(e_i) = c_i$. For v_0, we set $b(v_0) = \infty$.

Consider a feasible b-matching in G. By the construction of G, every edge in E corresponds to a word in $L(\langle \mathcal{A}, c\rangle) \cap L$. The budget constraints on the vertices correspond to the edge-capacities. Thus, every set of words from L that can be mutually accepted by $\langle \mathcal{A}, c\rangle$ corresponds to a feasible b-matching in G and vice-versa. In particular, a maximum b-matching corresponds to an optimal utilization multiset that is contained in L. ◀

Next, we show that the above is the most positive result we can achieve for the max restricted-utilization problem. That is, when L may include three-letter words, the max-utilization problem becomes APX-hard. That is, there exists a constant c such that it is NP-hard to find an approximation algorithm with approximation ratio better than c.

▶ **Theorem 9.** *The max restricted-utilization problem is APX-hard. This is valid already when the restricting language consists only of words of length* 3 *and a unit capacity DCA.*

Proof. We show an approximation-preserving reduction from the maximum 3-bounded 3-dimensional matching problem (3DM-3). The input to the 3DM-3 problem is a set of triplets $T \subseteq X \times Y \times Z$, where $|X| = |Y| = |Z| = n$. The number of occurrences of every element of

$X \cup Y \cup Z$ in T is at most 3. The number of triplets is $|T| \geq n$. The desired output is a 3-dimensional matching in T of maximal cardinality; i.e., a subset $T' \subseteq T$, such that every element in $X \cup Y \cup Z$ appears at most once in T', and $|T'|$ is maximal. Kann showed in [14] that 3DM-3 is APX-hard.

Given an instance of 3DM-3, we construct a DCA $\langle \mathcal{A}, c \rangle$, with $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F \rangle$, as follows. First, $\Sigma = X \cup Y \cup Z, Q = \{q_0, q_1, q_2, q_3\}, Q_0 = \{q_0\}, F = \{q_3\}$. The transition relation Δ consists of $3n$ transitions all having capacity 1. There are n parallel transitions $\langle q_0, x_i, q_1 \rangle$ for all $1 \leq i \leq n$, n parallel transitions $\langle q_1, y_j, q_2 \rangle$ for all $1 \leq j \leq n$, and n parallel transitions $\langle q_2, z_k, q_3 \rangle$ for all $1 \leq k \leq n$. The capacity of all transitions is 1.

To complete the reduction, we define the restricting language $L = \{x_i \cdot y_j \cdot z_k : \langle x_i, y_j, z_k \rangle \in T\}$. Note that the reduction is polynomial. Also, since the 3DM instance is 3-bounded, we have that $|L| = O(n)$.

Let $W \subseteq L$ be a set of words that can be mutually accepted by $\mathcal{A}$. The unit capacities imply that every element in $X \cup Y \cup Z$ appears in at most one word in W. Thus, every matching in T corresponds to a possible utilization multiset W. In particular, a maximum matching corresponds to an optimal utilization multiset for $\langle \mathcal{A}, c \rangle$, contained in L. ◀

▶ Remark. In the *weighted max (possibly restricted) utilization* problem, there is a profit function $p : \Sigma^* \to \mathbb{R}$ that associates profit with each word. The profit function can be given, for example, by a weighted automaton. The optimal utilization set is now a set $W \subseteq \Sigma^*$ that can be mutually accepted by $\langle \mathcal{A}, c \rangle$ and for which $\sum_{w \in W} p(w)$ is maximal. In the restricted variant, we require $W \subseteq L$.

Clearly, the lower bounds we prove apply also to the weighted version. As we now show, the polynomial upper bound for the case of a restricting language that consists only of words of length at most 2 can be extended to the weighted setting. For that, we also need a weighted version of the b-matching problem. There, every edge $e \in E$ is associated with a profit $p(e)$, and the maximum b-matching problem is to find a matching that maximizes $\sum_e p(e)x_e$. The algorithm in [2] applies also to the weighted variant.

Now, in the algorithm described in the proof of Theorem 8, we define the profits as follows. For every edge $e \in E$, we set $p(e)$ to be the profit $p(w)$ of the corresponding word. If e corresponds to two words, w_1, w_2, then set $p(w) = \max\{p(w_1, p(w_2)\}$. ◀

References

1 B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

2 R. P. Anstee. A polynomial algorithm for b-matching: An alternative approach. *Information Processing Letters*, 24:153–157, 1987.

3 G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. In *Proc. 17th Int. Conf. on Foundations of Software Science and Computation Structures*, LNCS 8412, pages 119–133. Springer, 2014.

4 G. Baier, E. Köhler, and M. Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.

5 B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. In *Proc. 8th Int. Conf. on Computer Aided Verification*, LNCS 1102, pages 1–12. Springer, 1996.

6 A. Bouajjani, P. Habermehl, and T. Vojnar. Verification of parametric concurrent systems with prioritised FIFO resource management. *Formal Methods in System Design*, 32(2):129–172, 2008.

7 M. Cadilhac, A. Finkel, and P. McKenzie. On the expressiveness of Parikh automata and related models. In *3rd Workshop on Non-Classical Models for Automata and Applications*, pages 103–119, 2011.

8 T. Colcombet, D. Kuperberg, and S. Lombardy. Regular temporal cost functions. In *Proc. 37th Int. Colloq. on Automata, Languages, and Programming*, LNCS 6199, pages 563–574, 2010.

9 H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *Proc. 10th Int. Conf. on Computer Aided Verification*, LNCS 1427, pages 268–279. Springer, 1998.

10 M. Droste, W. Kuich, and H. Vogler (eds.). *Handbook of Weighted Automata.* Springer, 2009.

11 L. R. Ford and D. R. Fulkerson. *Flows in networks.* Princeton Univ. Press, Princeton, 1962.

12 J. E. Hopcroft and R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (2nd Edition).* Addison-Wesley, 2000.

13 O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.

14 V. Kann. Maximum bounded 3-dimensional matching is max-snp-complete. *Information Processing Letters*, 37(1):27–35, 1991.

15 F. Klaedtke and H. Rueß. Parikh automata and monadic second-order logics with linear cardinality constraints. Technical Report 177, Universität Freiburg, 2002.

16 F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In *Proc. 30th Int. Colloq. on Automata, Languages, and Programming*, LNCS 2719, pages 681–696. Springer, 2003.

17 A. R. Mahjoub and S. T. McCormick. Max flow and min cut with bounded-length paths: complexity, algorithms, and approximation. *Mathematical Programming*, 124(1-2):271–284, 2010.

18 R. G. Qiu and S. B. Joshi. Deterministic finite capacity automata: a solution to reduce the complexity of modeling and control of automated manufacturing systems. In *Proc. Symp. on Computer-Aided Control System Design*, pages 218 –223, 1996.

19 M. O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.

20 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

Algorithms, Games, and Evolution*

Erick Chastain[1], Adi Livnat[2], Christos H. Papadimitriou[3], and Umesh V. Vazirani[4]

1 **Computer Science Department, Rutgers University**
erick@cs.rutgers.edu
2 **Department of Biological Sciences, Virginia Tech**
Blacksburg, VA 24061, USA
adi@vt.edu
3 **Computer Science Division, University of California, Berkeley**
Berkeley, CA, 94720, USA
christos@cs.berkeley.edu
4 **Computer Science Division, University of California, Berkeley**
Berkeley, CA, 94720, USA
vazirani@cs.berkeley.edu

Abstract

Even the most seasoned students of evolution, starting with Darwin himself [1], have occasionally expressed amazement at the fact that the mechanism of natural selection has produced the whole of Life as we see it around us. From a computational perspective, it is natural to marvel at evolution's solution to the problems of robotics, vision and theorem proving! What, then, is the complexity of evolution, viewed as an algorithm? One answer to this question is 10^{12}, roughly the number of sequential steps or generations from the earliest single celled creatures to today's Homo Sapiens. To put this into perspective, the processor of a modern cell phone can perform 10^{12} steps in less than an hour. Another answer is 10^{30}, the degree of parallelism, roughly the maximum number of organisms living on the Earth at any time. Perhaps the answer should be the product of the two numbers, roughly 10^{42}, to reflect the total work done by evolution, viewed as a parallel algorithm.

Here we argue, interpreting our recently published paper [2], that none of the above answers is really correct. Viewing evolution as an algorithm poses an additional challenge: recombination. Even if evolution succeeds in producing a particularly good solution (a highly fit individual), its offspring would only inherit half its genes, and therefore appear unlikely to be a good solution. This is the core of the problem of explaining the role of sex in evolution, known as the "queen of problems in evolutionary biology" [3].

The starting point of [2] is the diffusion-equation-based approach of theoretical population geneticists [4], who analyze the changing allele frequencies (over the generations) in the gene pool, consisting of the aggregate of the genetic variants (or "alleles") over all genes (or "loci") and over all individuals in a species. Taking this viewpoint to its logical conclusion, rather than acting on individuals or species or genes, evolution acts on this gene pool, or genetic soup, by making it more "potent", in the sense that it increases the expected fitness of genotype drawn randomly from this soup. Moreover, for much genetic variation [5], this soup may be assumed to be in the regime of weak selection, a regime where the probability of occurrence of a certain genotype involving various alleles at different loci is simply the product of the probabilities of each of its alleles. In this regime, we show in [2] that evolution in the regime of weak selection can be formulated as a game, where the recombining loci are the players, the alleles in those loci are possible moves or actions of each player, and the expected payoff of each player-locus is

* This work was partially supported by National Science Foundation Grant CCF-1064785, National Science Foundation Grant CCF-0964033, Templeton Foundation Grant 39966, and National Science Foundation Grant CCR-0905626.

precisely the organism's expected fitness across the genotypes that are present in the population. Moreover, the dynamics specified by the diffusion equations of theoretical population geneticists is closely approximated by the dynamics of multiplicative weight updates (MWUA) [6].

The algorithmic connection to MWUA brings with it new insights for evolutionary biology, specifically, into the question of how genetic diversity is maintained in the presence of natural selection. For this it is useful to consider a dual view of MWUA [7], which expresses "what each gene is optimizing" as it plays the game. Remarkably this turns out to be a particular convex combination of the entropy of its distribution over alleles and cumulative expected fitness. This sheds new light on the maintenance of diversity in evolution.

All of this suggests that the complexity of evolution should indeed be viewed as 10^{12}, but for a subtle reason. It is the number of steps of multiplicative weight updates carried out on allele frequencies in the genetic soup. A closer examination of this reveals further that the accurate tracking of allele frequencies over the generations requires the simulation of a quadratic dynamical system (two parents for each offspring). Moreover the simulation of even simple quadratic dynamical systems is known to be PSPACE-hard [8]. This suggests that the tracking of allele frequencies might require large population sizes for each species, putting into perspective the number 10^{30}. Finally, it is worth noting that in this view there is a primacy to recombination or sex, which serve to provide robustness to the mechanism of evolution, as well as the framework within which MWUA operates.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases evolution, recombination, coordination games, multiplicative weight updates

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.45

Category Invited Talk

References

1 Darwin, F. ed. 1887. The life and letters of Charles Darwin, including an autobiographical chapter. Vol. 2. London: John Murray.

2 Chastain, E., Livnat, A., Papadimitriou, C., Vazirani, U. "Algorithms, Games, and Evolution," PNAS, vol. 111, no. 29, 10620–10623, 2014.

3 Bell, G. 1982. The Masterpiece of Nature: the Evolution and Genetics of Sexuality. University of California Press, Berkeley.

4 Kimura, M. 1985. The Neutral Theory of Molecular Evolution. Cambridge University Press.

5 Nei, M. 2005. "Selectionism and neutralism in molecular evolution," Mol. Biol. Evol., 22:2318–2342.

6 Arora, S., Hazan E., Kale S. "The Multiplicative Weights Update Method: A Meta-Algorithm and Applications," *Theory of Computing*, 8(1): 121–164, 2012.

7 Orecchia L., Mahoney M. "Implementing Regularization Implicitly Via Approximate Eigenvector Computation," Proc. 28th Int'l Conf. Machine Learning, pp. 121–128, 2011.

8 Arora, S., Rabani, Y., Vazirani, U., "Simulating quadratic dynamical systems is PSPACE-complete," Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 459–467, 1994.

The Polynomial Method in Circuit Complexity Applied to Algorithm Design*

Richard Ryan Williams

Computer Science Department, Stanford University
353 Serra Mall, Stanford, CA 94305, USA
`rrw@cs.stanford.edu`

Abstract

In circuit complexity, the *polynomial method* is a general approach to proving circuit lower bounds in restricted settings. One shows that functions computed by sufficiently restricted circuits are "correlated" in some way with a low-complexity polynomial, where complexity may be measured by the degree of the polynomial or the number of monomials. Then, results limiting the capabilities of low-complexity polynomials are extended to the restricted circuits.

Old theorems proved by this method have recently found interesting applications to the design of algorithms for basic problems in the theory of computing. This paper surveys some of these applications, and gives a few new ones.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases algorithm design, circuit complexity, polynomial method

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.47

Category Invited Talk

1 Introduction

The polynomial method was developed for proving impossibility results on the capabilities of "low-complexity" circuits. (The usual measures of "low-complexity" for circuits are either low size, or low depth, or both.) The idea of the polynomial method, at a high level, is to show that functions computable by low-complexity circuits can be also be computed (approximately or exactly) by a "low-complexity" polynomial over some algebraic structure. Typically, the complexity of a polynomial is measured by its degree, but the complexity could also be the number of monomials. The survey by Beigel [8] contains many references to papers in which low-complexity circuits are represented via low-complexity polynomials, resulting in lower bounds against those circuits.

While the method was initially conceived to show the limitations of computational devices, the intermediate theorems proved via the method turn out to also be rather useful in the design of algorithms for certain problems – *positive* results about computational devices. Over the last few years, we have found some unexpected applications of the polynomial method to developing more efficient algorithms for several fundamental computational problems. Sometimes it is natural to see how the polynomial method might help; in other cases, it is not at all obvious, and some ingenuity is required. An intuitive outline of the approach is:

1. Find a "hard part" of one's computational problem that can be modeled by low-complexity circuits.

* This work was partially supported by the National Science Foundation Grant CCF 1212372.

2. Apply the polynomial method to convert the low-complexity circuits into an algebraic, polynomial form.
3. Use other algebraic algorithms to efficiently manipulate or evaluate these polynomials, thereby solving the original “hard part” more efficiently.

This article outlines several instances of this approach. Some of the algorithms in this paper are new to the literature; they are included to illustrate the versatility of the polynomial method in algorithm design. Many proofs of the known results are omitted from this article; however, some results stated here are new, and we shall describe their proofs in detail.

1.1 What This Survey is NOT

We make no claims to be the “first” to apply the polynomial method in positive algorithmic ways. There are many theorems in mathematics and theoretical computer science regarding the modeling of efficient functions with polynomials; discussing all of them is neither wise nor possible in this space. Nevertheless, it’s important to note that there are more interesting theorems related to the polynomial method, in the hopes that future work will make use of them. To give three examples from different angles:

1. Many theorems from approximation theory (which is effectively the study of point-wise approximating functions via “simple” expressions, such as polynomials over the reals) have seen applications in areas such as communication complexity and quantum computing [31, 7, 1]. We haven’t yet personally found algorithmic applications of these polynomials for our problems of interest, but that is probably our own failing, and not that of the polynomials.
2. Another example is the collection of lemmas in the literature informally known as the Schwartz-Zippel-DeMillo-Lipton Lemma [35, 50, 16] concerning the (low) number of zeroes in low-degree polynomials that are not identically zero. These lemmas are already a staple of randomized algorithms [30].
3. The polynomial method has found a large number of applications in *computational learning theory*, such as in algorithms for learning DNFs and low-depth circuits (e.g., [26, 27, 24, 19]) and learning functions with a small number of relevant variables (a.k.a. juntas) [29].

2 The Circuits

We assume the reader is familiar with the usual notion of Boolean circuits as directed acyclic graphs, where n input gates are represented by $2n$ source nodes (the n input bits and their negations), the output gate is represented by a single sink node, and each node (or “gate”) is labeled with a boolean function. We shall consider two well-studied restrictions of this general notion. Let $d, m \in \mathbb{N}$.

- An AC circuit of *depth* d is such that the longest path from any source to sink is at most d, and each gate computes either the OR (of its inputs) or the AND (of its inputs).
- An ACC circuit of *depth* d and *modulus* m is such that the longest path from any source to sink is at most d, and each gate computes one of OR, AND, or MODm, where $MOD\text{m}(y_1, \ldots, y_t) = 1$ if and only if $\sum_i y_i$ is divisible by m.

  More background can be found in the textbooks [5, 42].

3 The Tools

In this survey, we shall focus on just a few polynomial constructions from the literature which have recently been helpful. Again, we have made no attempt to be comprehensive.

Three notions of representation by polynomials will be considered in this article: exact representations, probabilistic representations over finite fields and the integers,

In the following, let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function with 0 representing *false* and 1 representing *true*, and let R be a ring containing 0 and 1. We shall always be evaluating polynomials over the values 0, 1, or -1. Since $x^a \in \{x, -x\}$ for all $a \in \mathbb{N}^+$ and all $x \in \{0, 1, -1\}$, it suffices for us to consider *multilinear* polynomials, of the form

$$p(x_1, \ldots, x_n) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i,$$

where $c_S \in R$ for all S. The *degree* of p is therefore the maximum cardinality of a subset S such that $c_S \neq 0$, and the *sparsity* of a polynomial is the number of S such that $c_S \neq 0$.

3.1 Exact Representations

▶ **Definition 1.** An n-variate polynomial $p(x_1, \ldots, x_n)$ over R *exactly represents* f if for all $(a_1, \ldots, a_n) \in \{0,1\}^n$, $p(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$.

When R is a field, every Boolean function f has a *unique* exact representation as a (multilinear) polynomial p. To give two simple examples, the function AND : $\{0,1\}^2 \to \{0,1\}$ is exactly represented by the polynomial $p(x_1, x_2) = x_1 x_2$, and OR : $\{0,1\}^2 \to \{0,1\}$ is represented by $p(x_1, x_2) = x_1 + x_2 - x_1 x_2$, over any field.

Exact representations are often implicitly used in algorithms, but their influence can be somewhat hidden. For example, the inclusion-exclusion principle from combinatorics can be applied to solve several hard problems more efficiently, e.g., counting the number of Hamiltonian Paths in n-node graphs in $2^n \cdot n^{O(1)}$ time and $n^{O(1)}$ space [23]. This principle is a consequence of the fact that the OR function on n variables can be exactly represented as:

$$\mathrm{OR}(x_1, \ldots, x_n) = 1 - \prod_{i=1}^{n} (1 - x_i) = \sum_{S \subseteq [n], |S| > 0} (-1)^{|S|+1} \prod_{i \in S} x_i.$$

In many situations, it is preferable to think of the Boolean function f with domain $\{-1, 1\}$ and codomain $\{-1, 1\}$ instead, where -1 corresponds to *true* and 1 corresponds to *false*. Then, a monomial $x_1 x_2 \cdots x_n$ represents the *PARITY* of n bits rather than the *AND* of n bits. Studying Boolean functions via this representation is often called the *Fourier analysis* of Boolean functions and is a world unto itself; we recommend O'Donnell's comprehensive textbook on the subject [32].

3.2 Probabilistic Representations

The next representation we consider is a "randomized" notion of polynomial, which is surprisingly powerful.

▶ **Definition 2.** Let $\mathcal{D}$ be a finite distribution of polynomials on n variables over R. The distribution $\mathcal{D}$ is a *probabilistic polynomial over R representing f with error δ* if for all $(a_1, \ldots, a_n) \in \{0,1\}^n$, $\Pr_{p \sim \mathcal{D}}[p(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)] > 1 - \delta$.

The *degree* of $\mathcal{D}$ is the maximum degree over all polynomials in $\mathcal{D}$.

One may also define a probabilistic polynomial as a *single* polynomial with n "input" variables and r "random" variables over a finite domain. Then, the distribution $\mathcal{D}$ in the above definition is obtained by assigning the r variables to uniform random values. However, it's not hard to see that, for every finite distribution $\mathcal{D}$ of s polynomials of maximum degree d and maximum sparsity m, one can recover a single probabilistic polynomial of degree d (in the input variables) with only $O(\log s)$ random variables and sparsity $O(m \cdot s)$, by simple interpolation (see also Tarui [40]).

Another important fact is that, (essentially) without loss of generality, the distribution $\mathcal{D}$ contains only $O(n)$ polynomials. Given any $\mathcal{D}$ for a function f and a parameter $\varepsilon > 0$, uniformly sample $t = O(n/\varepsilon^2)$ polynomials $p_1, \ldots, p_t \sim \mathcal{D}$, and form the distribution $\mathcal{D}'$ over $\{p_1, \ldots, p_t\}$ (as a multiset). By a standard Chernoff bound and union bound argument, the distribution $\mathcal{D}'$ is also a probabilistic polynomial for f, with essentially the same error (to within $\pm\varepsilon$).

Probabilistic polynomials were first utilized by Razborov [33] and Smolensky [36] in their proofs that the MAJORITY function and MOD3 functions cannot be computed efficiently with ACC circuits of constant depth and modulus 2, respectively. In particular, they showed that every low-depth circuit with modulus 2 has a low-degree probabilistic representation over the field $\mathbb{F}_2$. Here, we cite a strengthened version by Kopparty and Srinivasan:

▶ **Theorem 3** ([36, 25]). *For every* ACC *circuit C of depth d, size s, modulus 2, and n inputs, and $\varepsilon > 0$, there is a probabilistic polynomial $\mathcal{D}_C$ over $\mathbb{F}_2$ representing C with error ε, and degree at most $(4 \log s)^{d-1} \cdot (\log 1/\varepsilon)$, such that a polynomial p can be sampled from $\mathcal{D}_C$ in $n^{O(\log s)^{d-1}(\log 1/\varepsilon)}$ time.*

The basic idea of the proof is to randomly replace each gate in the circuit with very low-degree polynomials over $\mathbb{F}_2$, such that their composition leads to a low-degree polynomial for the entire circuit C. (The proof of Theorem 3 gives a clever way of composing these polynomials so as to keep the degree low, as a function of ε.) How do we construct these very low-degree polynomials? Gates which are MOD2 functions are simply additions over $\mathbb{F}_2$. A gate g which is a NOT of a gate h can be written as $g = 1 + h$ over $\mathbb{F}_2$. Gates which are ANDs can be expressed with NOTs and ORs by DeMorgan's law. Finally, gates which are ORs can be simulated probabilistically by multiplying a few sums of random subsets of the inputs, modulo 2. For example, if the OR of $x_1, \ldots, x_n$ is 1, then

$$\Pr_{r_1,\ldots,r_n \in \{0,1\}} \left[\sum_{i=1}^{n} r_i x_i = 1 \bmod 2 \right] = \frac{1}{2}.$$

On the other hand, if $x_1 = \cdots = x_n = 0$, then no random sum of the x_i's will evaluate to 1. In this way, a MOD2 can simulate an OR; multiplying several copies of such a probabilistic polynomial (carefully) will allow us to reduce the probability of error.

The above ideas can be extended to any finite field; however, the degrees of the probabilistic polynomials obtained may increase as a function of the field characteristic. (In particular, sums of variables will need to be raised to their $(p-1)$th powers, to keep the output Boolean.) It is natural to then ask how probabilistic polynomials over $\mathbb{Z}$ fare in computing AC circuits.

Beigel, Reingold, and Spielman [9] addressed this question, finding an $O(\log^2 n)$-degree probabilistic polynomial for OR. The following improvement is due to Aspnes, Beigel, Furst, and Rudich [6]:

▶ **Theorem 4.** *For all $\varepsilon > 0$, there is a probabilistic polynomial over $\mathbb{Z}$ for the OR of n variables with error ε, and degree $O(\log n \cdot \log 1/\varepsilon)$. Furthermore, for every* AC *circuit C of depth d and size s, there is a probabilistic polynomial for C with error ε having degree $O(\log^d s \cdot \log^d s/\varepsilon)$.*

Let us sketch the proof. To compute the OR of $x_1, \ldots, x_n$, choose progressively smaller random subsets $S_0, \ldots, S_{\log n+1} \subseteq \{1, \ldots, n\}$, where $S_0 = [n]$, and S_i is a uniform random subset of S_{i-1}. The key claim is that, if the OR of $x_1, \ldots, x_n$ is 1, then with probability at least 1/3, some S_i contains *exactly one* j such that $x_j = 1$. In that case, the polynomial

$$p(x_1, \ldots, x_n) = \prod_{i=0}^{\log n+1} \left(1 - \sum_{j \in S_i} x_j\right)$$

correctly computes the negation of OR (so, $1 - p$ computes OR). To reduce the error to arbitrarily small ε, one can take $O(\log 1/\varepsilon)$ products of independent copies of p.

To get probabilistic polynomials for AC circuits of depth d and size s with error ε, apply this randomized construction of p (and an analogous construction for AND) independently to every gate in the AC circuit, with error parameter set to ε/s. Then, a union bound over all s gates guarantees the result.

3.3 Symmetric Representation

Finally, we consider a polynomial representation of functions which may look somewhat unusual: we try to represent functions by low-complexity polynomials h whose outputs are "filtered" through another function g which gives $\{0, 1\}$ output.

▶ **Definition 5.** Let $h(x_1, \ldots, x_n)$ be a polynomial over R, construed as a function $h : \{0, 1\}^n \to R$. Let $g : \text{Im}(h) \to \{0, 1\}$ be arbitrary. We say that (g, h) is a *symmetric representation of* f if for all $(a_1, \ldots, a_n) \in \{0, 1\}^n$, $g(h(a_1, \ldots, a_n)) = f(a_1, \ldots, a_n)$.

Why do we call this a "symmetric" representation? Suppose $R = \mathbb{Z}$. If all coefficients of h are in $\{0, 1\}$ and h has s monomials, we have $\text{Im}(h) \subseteq \{0, 1, \ldots, s\}$, and the "filter function" g may then be viewed as a function on s variables which only depends on the number of inputs which are true. That is, we may think of g as a *symmetric* Boolean function. To put it another way, in this situation we can represent $g \circ h$ as a depth-two Boolean circuit with $s + 1$ gates, where the output gate computes a symmetric function and the layer of gates nearest the inputs compute ANDs. (The function h counts up the number of ANDs which output true, and the function g determines the output of the symmetric function.)

Symmetric representations are not as unusual as one might think. The class of *polynomial threshold functions* refer to a particular type of symmetric representation, where the symmetric function is a threshold function (checking whether the sum of all inputs exceeds a fixed value T). Polynomial threshold functions have been studied for a rather long time, especially in the context of neural networks ([28]).

We will use a particularly strong result on symmetric representations of functions computable with ACC circuits, first proved by Beigel and Tarui, building on work of Yao:

▶ **Theorem 6** (Yao [49], Beigel and Tarui [10])**.** *There is a function $\alpha : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that every Boolean function computable by an* ACC *circuit with size s, depth d, and modulus m has a symmetric representation (g, h) with $\deg(h) \leq (\log s)^{\alpha(d,m)}$.*

That is, every constant-depth and constant-modulus circuit can be symmetrically represented with a polynomial of degree that is polylogarithmic in the circuit size. It is widely believed that this sort of representation should severely restrict the kinds of functions computable with constant depth and modulus. It is believed that the MAJORITY of n bits cannot be computed with polynomial-size ACC circuits of any constant depth or constant modulus.

4 The Applications

Now we discuss how these tools have been recently applied in algorithm design.

4.1 All-Pairs Shortest Paths (APSP)

We first study the dense case of the All-Pairs Shortest Paths problem (APSP) on general weighted graphs.

▶ **Definition 7** (All-Pairs Shortest Paths (APSP)). In *APSP*, the input is a weighted adjacency matrix, and the goal is to output a data structure S encoding all shortest paths between any pair of nodes: when a pair of nodes (s, t) are fed to S, it must reply with the shortest distance from s to t in $\tilde{O}(1)$ time, and an actual shortest path from s to t in $\tilde{O}(\ell)$ time, where ℓ is the number of edges on the path.

The $O(n^3)$ time algorithms for APSP on n-node graphs [17, 43] are in the canon of undergraduate computer science. But these algorithms could be suboptimal, as the input graph can be encoded in $\Theta(n^2 \cdot \log m)$ bits, where $m \in \mathbb{N}$ upper bounds the edge weights. Indeed, in the real RAM model of computation (where additions and comparisons on "real-valued" registers are allowed, and arbitrary bit operations on "word registers" of $O(\log n)$ bits), Fredman [18] showed in 1975 that APSP is solvable in $o(n^3)$ time.

Since then, many papers on the dense case of APSP have been published, steadily decreasing the running time of $O(n^3)$ [37, 38, 51, 20, 39, 21, 13, 14]. All of them obtained only $O(n^3/\log^c n)$ time algorithms for constants $c \leq 2$. A major open problem is whether APSP is solvable in truly subcubic time, i.e., $O(n^{3-\varepsilon})$ time for some fixed $\varepsilon > 0$. A recently developed hardness theory for APSP shows that such an algorithm would have many consequences [34, 41, 3, 2].

Recently, the author gave new algorithms for APSP that run faster than $O(n^3/\log^c n)$ time, for *every* constant c [45]. In fact, the randomized version runs in $n^3/2^{\Omega(\sqrt{\log n})}$ time and the deterministic version runs in $n^3/2^{\Omega((\log n)^\delta)}$ time for some $\delta > 0$; these are asymptotically much better bounds. The algorithms crucially rely on the tools of the previous section: the problem of efficiently computing APSP is reduced to efficiently computing a particular circuit evaluation problem, and it is shown how to evaluate such circuits more efficiently than the obvious approach.

Let us jump directly to the kind of circuit that arises from the proof. Implicit in the APSP paper [45] is the following theorem, which we isolate for the reader's convenience. For $d, n \in \mathbb{N}$, define the Boolean function OR-AND-COMP on $2d^2 \log n$ inputs as follows:

$$\text{OR-AND-COMP}(a_{1,1}, a_{1,2}, \ldots, a_{d,d}, b_{1,1}, b_{1,2}, \ldots, b_{d,d}) := \bigvee_{i=1}^{d} \bigwedge_{j=1}^{d} [a_{i,j} \leq b_{i,j}],$$

where the $a_{i,j}, b_{i,j}$ are construed as $\log n$-bit numbers and $[A \leq B]$ is true if and only if $A \leq B$.

▶ **Theorem 8** (Implicit in [45]). *Let A, B be two sets of n vectors, where each vector is of length d^2 and each vector component has $\log n$ bits. Suppose the function OR-AND-COMP$(a_1, \ldots, a_{d^2}, b_1, \ldots, b_{d^2})$ is computable in $\tilde{O}(n^2)$ time, for all n^2 pairs $(a_1, \ldots, a_{d^2}) \in A$ and $(b_1, \ldots, b_{d^2}) \in B$ simultaneously. Then APSP is solvable in $\tilde{O}(n^3/d)$ time.*

Why is this theorem true? Here's a little intuition. APSP involves comparing the sums of weights on different paths, and determining which sums of weights are minimal among a collection of sums. The OR-AND-COMP circuit is effectively finding a minimum sum among a particular set of paths of length two. The ability to compute this minimum sum for all n^2 pairs of vectors roughly corresponds to computing APSP in a tripartite graph with n nodes in the first part, d nodes in a middle part, and n nodes in the third part, with first and third part disconnected. Of course this is extremely handwavy, and the reader should consult the paper for more details.

As the circuit OR-AND-COMP has $2d^2 \log n$ inputs, such an evaluation would naively take $O(n^2 d^2 \log n)$ time. Presumably, it is easier to get an $\tilde{O}(n^2)$ time algorithm when d is small. The APSP paper [45] shows that for $d = 2^{c\sqrt{\log n}}$ where $c > 0$ is some constant, the $\tilde{O}(n^2)$ time evaluation required by Theorem 8 is actually possible. Here's a high-level outline of the algorithm.

1. First, computing $[a \le b]$ for $(\log n)$-bit strings a and b can be done with constant-depth $O((\log n)^2)$-size circuits over AND and OR; that is, OR-AND-COMP is computable with AC circuits of constant depth and polynomial size. So the first idea is to apply Razborov and Smolensky (Theorem 3) (or Beigel-Tarui, Theorem 6) to the AC circuit for OR-AND-COMP, reducing this circuit to a probabilistic polynomial (or a symmetric representation, respectively). Given that this AC circuit has size $O(d^2 \log^2 n)$ on $O(d^2 \log n)$ variables, we find that OR-AND-COMP has a probabilistic polynomial over $\mathbb{F}_2$ with $2^{\text{poly}(\log d, \log\log n)}$ monomials and $< 1/n^3$ error, and there is a symmetric representation of OR-AND-COMP with a similar monomial upper bound.
2. Second, given two sets A, B of n vectors as in the theorem statement, we show how to efficiently evaluate polynomials with at most $n^{.1}$ monomials on all pairs of vectors (one from A and one from B). This step uses a special rectangular matrix multiplication algorithm of Coppersmith [15], and runs in $\tilde{O}(n^2)$ time.
3. Thirdly, we combine 1 and 2. We use part 1 to generate a polynomial representation for OR-AND-COMP with $m = 2^{\text{poly}(\log d, \log\log n)}$ monomials. Choose $d = 2^{(\log n)^{\delta}}$, so that $\delta > 0$ is small enough to make $m \le n^{.1}$. Now we can apply part 2 and compute this OR-AND-COMP (with some probability of error) in $\tilde{O}(n^2)$ time. If our polynomials are probabilistic, each evaluation may have some errors. However, if we take $O(\log n)$ independent constructions and evaluations of these probabilistic polynomials for OR-AND-COMP, the MAJORITY values of these $O(\log n)$ evaluations will yield the correct values for OR-AND-COMP on all n^2 pairs of points, with high probability. Finally, applying Theorem 8, we thereby compute APSP in $n^3/2^{\Omega((\log n)^{\delta})}$ time.

With a few pages of technical work, the $\delta > 0$ in the algorithm can be tuned down to $1/2$ in the randomized case. The big question is whether we can set $\delta = 1$, and yield a truly-subcubic APSP algorithm. This looks difficult, and not just because it is a thorny circuit evaluation problem.

If we try to use Theorem 8 to get truly-subcubic APSP, we would need a fast algorithm for evaluating OR-AND-COMP with $d \ge n^{\delta}$ for some $\delta < 1$. However, such an algorithm would also resolve *another* major open problem: we'd be able to solve CNF-SAT in $2^{\delta n}$ time for some $\delta < 1$, contradicting the so-called Strong Exponential Time Hypothesis (SETH) [22, 12]. In the following section, we shall explain why.

4.2 Orthogonal Vectors (OV)

We consider a slightly simpler function than the one needed for solving APSP. Define

$$\text{OR-AND-OR2}_{d_1,d_2}(x_{1,1}, x_{1,2}, \ldots, x_{d_1,d_2}, y_{1,1}, y_{1,2}, \ldots, y_{d_1,d_2}) = \bigvee_{i=1}^{d_1} \bigwedge_{j=1}^{d_2} (x_{i,j} \vee y_{i,j}).$$

That is, OR-AND-OR2$_{d_1,d_2}$ takes $2d_1d_2$ bits of input. Note that we can easily simulate OR-AND-OR2$_{d,d}$ with a call to OR-AND-COMP: $(x_{i,j} \vee y_{i,j}) = 1$ if and only if $[\neg x_{i,j} \leq y_{i,j}]$. Therefore, evaluating OR-AND-OR2$_{d,d}$ is only easier than evaluating OR-AND-COMP. However, quick evaluation of OR-AND-OR2 would yield faster algorithms for other problems than just APSP. Here is the canonical example of such a problem:

▶ **Definition 9** (Orthogonal Vectors (OV)). In *OV*, the input is a set $S \subseteq \{0,1\}^d$, and the goal is to output whether there are vectors $a, b \in S$ such that $\langle a, b\rangle = 0$.

That is, we wish to know if S contains an orthogonal pair of vectors. There are two obvious algorithms: one takes $O(|S|^2 d)$ time, and one takes $O(2^d|S|)$ time. So the interesting case is when we have "high dimensionality", and $d \geq \log n$. It is an open question whether $O(|S|^{2-\varepsilon}2^{o(d)})$ time is possible for some fixed $\varepsilon > 0$. By adding two more dimensions to the vectors, the following version of OV is equivalent to the above:

▶ **Definition 10** (Orthogonal Vectors' (OV')). In *OV'*, the input is two sets $A, B \subseteq \{0,1\}^d$, and the goal is to output whether there are vectors $a \in A$, $b \in B$ such that $\langle a, b\rangle = 0$.

OV captures the difficulty of several problems. Consider the partial match problem from string searching:

▶ **Definition 11** (Batch Partial Match (BPM)). In *BPM*, the input is a database $D \subseteq \{0,1\}^d$, and queries $Q \subseteq \{0,1,\star\}^d$, where $|D| = |Q|$. The goal is to output, for every $q \in Q$, whether or not there is an $x \in D$ such that for all $i = 1,\ldots,d$, $q[i] \neq \star$ implies $q[i] = x[i]$.

That is, we wish to know which queries have a "partial match" in the given database. Recent work with Abboud and Yu [4] proved that BPM is *sub-quadratic equivalent* to OV: roughly speaking, an $|S|^{2-\varepsilon}f(d)$ time algorithm for OV implies an $|Q|^{2-\delta}f(d)$ time algorithm for the BPM, and the converse also holds.

Another string problem related to OV is a generalization of the longest common substring problem to handle wildcard symbols:

▶ **Definition 12** (Longest Common Substring With Don't Cares (LCS*)). In *LCS**, the input is two strings $S, T \in \Sigma^n$ of length n, and the goal is to output the length of the longest string that appears in both S and T as a contiguous substring.

In the same paper with Abboud and Yu, it is proven that LCS* has a faster-than-quadratic time algorithm, given that OV has one. The importance of solving OV in sub-quadratic time is further reinforced by the following connection with exponential-time algorithms for satisfiability.

▶ Conjecture 4.1 (Strong Exponential Time Hypothesis (SETH) [22, 12]). For every $\delta < 1$, there is a $k \geq 3$ such that satisfiability of k-CNF formulas on n variables requires more than $2^{\delta n}$ time.

▶ **Theorem 13** ([44, 48]). *Suppose there is an $\varepsilon > 0$ such that for all $c \geq 1$, OV can be solved in $O(|S|^{2-\varepsilon})$ time on instances with $c \log |S|$ dimensions. Then SETH is false.*

Proof. We prove the contrapositive. Calabro, Impagliazzo, and Paturi [11] show that refuting SETH is equivalent to giving a $\delta < 1$ such that, for all $c \geq 1$, CNF-SAT on instances with n variables and cn clauses can be solved in $O(2^{\delta n})$ time.

We reduce this variant of CNF-SAT to OV. Given a formula F on n variables and cn clauses $C_1, \ldots, C_{cn}$, divide the variables into two sets V_1 and V_2 with at most $n/2 + 1$

variables each. Enumerate all $O(2^{n/2})$ partial assignments to the variables in V_1 and all partial assignments to the variables in V_2. For each such partial assignment A, define a vector v_A with $cn+2$ dimensions as follows. For $i=1,\ldots,cn$, set $v_A[i]=0$ iff the clause C_i is satisfied by A. Then, set $v_A[cn+1]=1$ iff the partial assignment A is on the variables of set V_1, and set $v_A[cn+2]=1$ iff A is from set V_2. Put all v_A's in the OV instance S.

Suppose F is satisfiable; let A be a satisfying assignment. For $i=1,2$, let the partial assignment A_i be the assignment A restricted to variables from V_i. By construction, $v_{A_1}[cn+1]\cdot v_{A_2}[cn+1]=v_{A_1}[cn+2]\cdot v_{A_2}[cn+2]=0$, and for every clause C_i, at least one of A_1 or A_2 satisfies C_i, so $v_{A_1}[i]\cdot v_{A_2}[i]=0$. It follows that $\langle v_{A_1},v_{A_2}\rangle=0$. Similarly, $\langle v_A,v_{A'}\rangle=0$ implies that A and A' come from different sets and jointly satisfy F.

Finally, if OV is in $O(|S|^{2-\varepsilon})$ time for $c\log|S|$ dimensional vectors, then we can determine satisfiability of F in $O(2^{n(1-\varepsilon/2)})$ time. ◀

Now we can formally illustrate the importance of evaluating OR-AND-OR2 efficiently:

▶ **Theorem 14** (Implicit in [4]). *Let A,B be two sets of n bit vectors, where each vector has $t=d_1\cdot d_2$ bits. Suppose OR-AND-OR2$_{d_1,d_2}(a_1,\ldots,a_t,b_1,\ldots,b_t)$ is computable in $\tilde{O}(n^2)$ time, for all n^2 pairs $(a_1,\ldots,a_t)\in A$ and $(b_1,\ldots,b_t)\in B$ simultaneously. Then OV with n vectors in d_2 dimensions can be solved in $\tilde{O}(n^2/d_1)$ time.*

Proof. For convenience, we work with OV' (Definition 10) in which we get two sets of vectors A,B and wish to find $a\in A$ and $b\in B$ that are orthogonal.

Partition both A and B into $\sqrt{d_1}$-size subsets $A_1,\ldots,A_{O(n/\sqrt{d_1})}$ and $B_1,\ldots,B_{O(n/\sqrt{d_1})}$, respectively. The idea is that with a single OR-AND-OR2$_{d_1,d_2}$ computation on $2d_1d_2$ bits, we can check whether the sub-instance (A_i,B_j) contains an orthogonal pair of vectors, for all $i,j=1,\ldots,\sqrt{d_1}$.

The function OR-AND-OR2$_{d_1,d_2}$ takes the OR over d_1 pairs of vectors of the *complement of the Boolean inner product of d_2-dimensional vectors.* That is, the AND-OR2 parts of the function output 1 if the two relevant d_2-dimensional vectors are orthogonal, and 0 otherwise. By arranging the $\sqrt{d_1}$ vectors of A_i into one d_1d_2-dimensional vector, and doing the same for B_j, we can check whether the $\sqrt{d_1}$-size set A_i and the $\sqrt{d_2}$-size set B_j contain an orthogonal pair with one call to OR-AND-OR2$_{d_1,d_2}$. There are several ways to do this. For example, if the vectors of A_i are $a_1,\ldots,a_{\sqrt{d_1}}$ and the vectors of B_j are $b_1,\ldots,b_{\sqrt{d_1}}$, then we may define the (d_1d_2)-dimensional vectors

$$v_{A_i}:=(a_1,\ldots,a_1,\ a_2,\ldots,a_2,\ \cdots\ ,a_{\sqrt{d_1}},\ldots,a_{\sqrt{d_1}}),$$

$$v_{B_i}:=(b_1,b_2,\ldots,b_{\sqrt{d_1}},\ b_1,b_2,\ldots,b_{\sqrt{d_1}},\ \cdots\ ,b_1,b_2,\ldots,b_{\sqrt{d_1}}),$$

where the '...' in the v_{A_i} denote $\sqrt{d_1}$ repetitions of the same vector. Then,

$$\text{OR-AND-OR2}_{d_1,d_2}(v_{A_i},v_{B_j})=0 \iff \text{there is no orthogonal pair in } (A_i,B_j).$$

Constructing the sets of vectors $A'=\{v_{A_1},\ldots,v_{A_{O(n/\sqrt{d_1})}}\}$ and $B'=\{v_{B_1},\ldots,v_{B_{O(n/\sqrt{d_1})}}\}$, we conclude that computing OR-AND-OR2$_{d_1,d_2}$ on all pairs of vectors in A' and B' will determine whether A,B has an orthogonal vector. By assumption, this computation can be done in $\tilde{O}((n/\sqrt{d_1})^2)\le\tilde{O}(n^2/d_1)$ time, which finishes the proof. ◀

▶ **Corollary 15.** *If there is an $\varepsilon>0$ such that for all $c\geq 1$ the hypothesis of Theorem 14 is true with $d_1\geq n^{\varepsilon}$ and $d_2\geq c\log n$, then SETH is false.*

Proof. Follows from combining Theorem 13 and Theorem 14. ◀

The above relations between OV and other problems show that finding orthogonal pairs of vectors is of importance. Recently, fast evaluation algorithms for OR-AND-OR2 have been developed, tailored to run faster than what's known for OR-AND-COMP (used to solve APSP in the previous section):

▶ **Theorem 16** (Implicit in Abboud, Williams, Yu [4]). *The function OR-AND-OR2$_{s,d}$ can be evaluated on two sets of n vectors in $\tilde{O}(n^2)$ time, provided that*

$$s^2 \cdot \binom{d+1}{3\log s}^2 \leq n^{0.1}.$$

The algorithm of Theorem 16 is obtained by converting OR-AND-OR2 into a probabilistic polynomial over $\mathbb{F}_2$ (via Theorem 3) and carefully counting the monomials that arise in the construction of the polynomial. In particular, each AND is converted into a $3\log s$-degree probabilistic polynomial with error less than $1/s^3$, and the topmost OR on s variables is converted into a product of two random MOD2s. After $O(\log n)$ evaluations of these probabilistic polynomials for OR-AND-OR2, we settle on the correct values for OR-AND-OR2 on all n^2 pairs of points. The fast $\tilde{O}(n^2)$ time evaluation is again done using Coppersmith's fast rectangular matrix multiplication [15].

The inequality of Theorem 16 holds for $s \leq n^{\varepsilon/\log(d/\log n)}$ where $\varepsilon > 0$ is sufficiently small. From this and the above theorems, we derive:

▶ **Corollary 17.**

- *OV on n vectors in d dimensions is in $n^{2-1/O(\log(d/\log n))}$ time.*
- *BPM on n strings of length d each is in $n^{2-1/O(\log(d/\log n))}$ time.*
- *LCS* on two strings of length n is in $n^2/2^{\Omega(\sqrt{\log n})}$ time.*
- *CNF-SAT on n variables and m clauses is solvable in $2^{n(1-1/O(\log(m/n))}$ time.*

For the first three problems, these running times significantly improve upon prior work. The running time stated for CNF-SAT is not new, but it does match (up to constant factors in the big-O) the best known CNF-SAT algorithms, which is fairly surprising given the generality of this approach.

4.3 Counting Solutions to OV and CNF-SAT

Applying probabilistic polynomials over $\mathbb{Z}$ instead of $\mathbb{F}_2$, we can count the number of solutions to an OV instance or a CNF formula. Let us remark that these results have not appeared in print before; while they are not significant extensions of the previous section, they should still give the reader a sense of what else is possible.

Define the function SUM-AND-OR2$_{d_1,d_2}$: $\{0,1\}^{d_1 \cdot d_2} \to \mathbb{N}$ as:

$$\text{SUM-AND-OR2}_{d_1,d_2}(x_{1,1}, x_{1,2}, \ldots, x_{d_1,d_2}, y_{1,1}, y_{1,2}, \ldots, y_{d_1,d_2}) = \sum_{i=1}^{d_1} \bigwedge_{j=1}^{d_2} (x_{i,j} \vee y_{i,j}).$$

That is, this function outputs the total sum (over the integers) of the true AND-OR2s.

▶ **Theorem 18.** *There is a probabilistic polynomial for SUM-AND-OR2$_{d_1,d_2}$ over $\mathbb{Z}$ with error at most $1/3$ and at most $(d_1)^{O(\log^2 d_2)}$ monomials.*

Proof. Think of the SUM-AND-OR2$_{d_1,d_2}$ as a circuit. Replace each of the d_1 ANDs of fan-in d_2 in this circuit with a probabilistic polynomial over $\mathbb{Z}$ with error set to $\varepsilon = 1/(3d_1)$. By Theorem 4, these polynomials have degree $O(\log d_2 \cdot \log d_1)$, and therefore they have at most $(d_2)^{O(\log d_2 \cdot \log d_1)}$ monomials, assuming the output of each OR2 gate is a variable in the polynomial. Now, each OR2 can be represented exactly as a sum of three monomials in the original variables, which means we obtain a polynomial with at most $(3d_2)^{O(\log d_2 \cdot \log d_1)} \leq (d_1)^{O(\log^2 d_2)}$ monomials in the original variables. Since each AND had error at most $1/(3d_1)$, their total sum is correct with probability at least $2/3$, by the union bound. ◀

Now, provided that d_1 and d_2 satisfy

$$(d_1)^{O(\log^2 d_2)} \leq n^{0.1},$$

the number of monomials is low, and we can apply the same strategy used in Theorem 14 to solve OV. Since we are taking a SUM instead of an OR, we can now compute the *number* of all orthogonal pairs in a set of d_2-vectors of size $O(\sqrt{d_1})$, in $\tilde{O}(n^2)$ time. The above inequality is certainly achieved when $d_1 \leq n^{1/O(\log^2 d_2)}$. Following the proof of Theorem 14 and computing the number of orthogonal pairs for all $O(n^2/d_1)$ pairs of sets, we obtain:

▶ **Theorem 19.** *The number of orthogonal pairs among n vectors in d dimensions is computable in $n^{2-1/O(\log^2 d)}$ time, with high probability. Consequently, one can count the number of matches in the database on a set of n BPM queries of length d in the same running time, and we can count the number of satisfying assignments to a CNF on n variables and m clauses in $2^{n(1-1/O(\log^2 m))}$ time.*

For counting OV pairs, the above running time is still much faster than $O(n^2/\text{poly}(\log n))$ when $d = \text{poly}(\log n)$. Indeed, it follows that counting the satisfying assignments of a CNF with n variables and $n^{\log n}$ clauses can be done in $2^{n-n/\text{poly}(\log n)}$ time.

5 Conclusion

We have seen several ways in which polynomial tools originally developed in circuit complexity have recently led to many new algorithms. We have not discussed all recent applications of the polynomial method: we've mostly ignored the (more obvious) application of the polynomial method for circuit lower bounds to solving *circuit satisfiability.* For example, the polynomial method tools discussed here also can be used to give faster algorithms for satisfiability of ACC circuits [47], as well as 0-1 linear programming [46] and satisfiability of symmetric Boolean CSPs [4].

We cannot help but point out a discrepancy between the usage of polynomials in circuit complexity and our algorithmic applications thus far. The majority of circuit lower bound results using polynomials focus on minimizing the *degree* of the polynomial representing the low-complexity function. However, for our applications, the *number of monomials*, or the sparsity, is the most important measure for our algorithmic applications. Certainly, a degree-d polynomial in n variables has $n^{O(d)}$ monomials, but this may be an undesirable representation for super-constant d. This survey shows that finding *sparse* polynomial representations for low-complexity functions like OR-AND-OR2 would entail significant algorithmic consequences.

Acknowledgements. I am grateful to Venkatesh Raman for suggesting the topic of this article, and his subsequent patience with me while I was finishing it.

References

1 Scott Aaronson. The polynomial method in quantum and classical computing. In *FOCS*, pages 3–3. IEEE, 2008.

2 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *SODA*, 2015.

3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, 2014.

4 Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *SODA*, page to appear, 2015.

5 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

6 James Aspnes, Richard Beigel, Merrick Furst, and Steven Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994.

7 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001.

8 Richard Beigel. The polynomial method in circuit complexity. In *In Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 82–95. IEEE Computer Society Press, 1995.

9 Richard Beigel, Nick Reingold, and Daniel Spielman. The perceptron strikes back. In *Structure in Complexity Theory Conference*, pages 286–291. IEEE, 1991.

10 Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, pages 350–366, 1994.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *IEEE Conf. Computational Complexity*, pages 252–260, 2006.

12 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

13 Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008. See also WADS'05.

14 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. See also STOC'07.

15 Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982.

16 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.

17 Robert W. Floyd. Algorithm 97. *Comm. ACM*, 5-6:345, 1962.

18 Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):49–60, 1976. See also FOCS'75.

19 Parikshit Gopalan and Rocco A. Servedio. Learning and lower bounds for AC0 with threshold gates. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 588–601. Springer, 2010.

20 Yijie Han. Improved algorithm for all pairs shortest paths. *Information Processing Letters*, 91(5):245–250, 2004.

21 Yijie Han. An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest path. *Algorithmica*, 51(4):428–434, 2008. See also ESA'06.

22 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

23 Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2):49–51, 1982.

24 Adam R. Klivans and Rocco Servedio. Learning DNF in time $2^{O(n^{1/3})}$. In *STOC*, pages 258–265. ACM, 2001.

25 Swastik Kopparty and Srikanth Srinivasan. Certifying polynomials for AC^0(parity) circuits, with applications. In *FSTTCS*, pages 36–47, 2012.

26 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.

27 Yishay Mansour. An $o(n^{\log \log n})$ learning algorithm for dnf under the uniform distribution. *Journal of Computer and System Sciences*, 50(3):543–550, 1995.

28 Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, 1969.

29 Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *Journal of Computer and System Sciences*, 69(3):421–434, 2004.

30 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

31 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994.

32 Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

33 A. A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

34 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *Algorithms–ESA 2004*, pages 580–591. Springer, 2004.

35 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

36 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

37 Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43(4):195–199, 1992. See also WG'91.

38 Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998. See also WG'95.

39 Tadao Takaoka. An $O(n^3 \log \log n / \log n)$ time algorithm for the all-pairs shortest path problem. *Information Processing Letters*, 96(5):155–161, 2005.

40 Jun Tarui. Probabilistic polynomials, AC0 functions and the polynomial-time hierarchy. *Theoretical computer science*, 113(1):167–183, 1993.

41 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654. IEEE, 2010.

42 Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer, 1999.

43 Stephen Warshall. A theorem on Boolean matrices. *J. ACM*, 9:11–12, 1962.

44 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. See also ICALP'04.

45 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673, 2014.

46 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *STOC*, pages 194–202, 2014.

47 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2, 2014.

48 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *SODA*, pages 1867–1877. SIAM, 2014.

49 Andrew Chi-Chih Yao. On ACC and threshold circuits. In *FOCS*, pages 619–627, 1990.

50 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Lecture Notes in Computer Science*, volume 72, pages 216–226. Springer, 1979.

51 Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In *ISAAC 2004*, volume 3341 of *Springer LNCS*, pages 921–932, 2004.

Vertex Exponential Algorithms for Connected f-Factors

Geevarghese Philip[1] **and M. S. Ramanujan**[2]

1 **Max-Planck-Institut für Informatik**
`gphilip@mpi-inf.mpg.de`
2 **University of Bergen**
`ramanujan.sridharan@ii.uib.no`

Abstract

Given a graph G and a function $f : V(G) \to [|V(G)|]$, an f-factor is a subgraph H of G such that $deg_H(v) = f(v)$ for every vertex $v \in V(G)$; we say that H is a *connected f-factor* if, in addition, the subgraph H is connected. Tutte (1954) showed that one can check whether a given graph has a specified f-factor in polynomial time. However, detecting a *connected* f-factor is NP-complete, even when f is a *constant* function – a foremost example is the problem of checking whether a graph has a Hamiltonian cycle; here f is a function which maps every vertex to 2. The current best algorithm for this latter problem is due to Björklund (FOCS 2010), and runs in randomized $\mathcal{O}^*(1.657^n)$ time (The $\mathcal{O}^*()$ notation hides polynomial factors). This was the first superpolynomial improvement, in nearly fifty years, over the previous best algorithm of Bellman, Held and Karp (1962) which checks for a Hamiltonian cycle in deterministic $\mathcal{O}(2^n n^2)$ time.

In this paper we present the first vertex-exponential algorithms for the more general problem of finding a connected f-factor. Our first result is a randomized algorithm which, given a graph G on n vertices and a function $f : V(G) \to [n]$, checks whether G has a connected f-factor in $\mathcal{O}^*(2^n)$ time. We then extend our result to the case when f is a mapping from $V(G)$ to $\{0, 1\}$ and the degree of every vertex v in the subgraph H is required to be $f(v)(mod\ 2)$. This generalizes the problem of checking whether a graph has an *Eulerian* subgraph; this is a connected subgraph whose degrees are all even ($f(v) \equiv 0$). Furthermore, we show that the *min-cost editing* and *edge-weighted* versions of these problems can be solved in randomized $\mathcal{O}^*(2^n)$ time as long as the costs/weights are bounded polynomially in n.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Exact Exponential Time Algorithms, f-Factors

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.61

1 Introduction

The problem of testing whether an input graph has a Hamiltonian cycle – a simple cycle which passes through all vertices of the graph – is one of Karp's original list of 21 NP-complete problems [13], and is one of the most fundamental and well-studied problems in computational complexity. The current best algorithm for this problem is due to Björklund (FOCS 2010), and runs in randomized $\mathcal{O}^*(1.657^n)$ time(The $\mathcal{O}^*()$ notation hides polynomial factors.) [2]. This was the first superpolynomial improvement in nearly fifty years, over the previous best algorithm of Bellman [1], and Held and Karp [12] which checks for a Hamiltonian cycle in deterministic $\mathcal{O}(2^n n^2)$ time.

Another fundamental graph problem is that of deciding whether a given graph contains a regular subgraph. This problem was first stated by Garey and Johnson [9] who asked if testing the presence of a 3-regular subgraph in a given graph is NP-complete. This was shown to be NP-complete in a proof atrributed to Chvátal [9]. However, testing if a graph has a *spanning* r-regular subgraph is known to be polynomial time solvable by an application of Tutte's f-factor theorem [20], although testing for a *connected* spanning r-regular subgraph clearly generalizes Hamiltonicity.

Given a graph G and a function $f : V(G) \to [|V(G)|]$, an f-factor of G is a subgraph H of G such that $deg_H(v) = f(v)$ for every vertex $v \in V(G)$; we say that H is a *connected f-factor* if, in addition, the subgraph H is connected. Tutte [20] showed that one can check whether a given graph has a specified f-factor in polynomial time. Lovász [14, 15] extended this result to *general* f-factors, where the function f maps each vertex to a list of numbers. Lovász and Cournéjols [4] gave a complete characterization of the complexity of the general f-factor problem.

In this paper we study the problem of finding *connected* f-factors in a given graph. Our main motivation in investigating this problem is the fact that it generalizes the problem of testing for a Hamiltonian cycle in a graph, and also the more general problem of testing for regular connected spanning subgraphs.

Our results and techniques. Although the existence of a connected f-factor in a graph withm m edges can trivially be tested in time $\mathcal{O}^*(2^m)$, it was not known whether it is possible to solve this problem in time which is single-exponential in the number of vertices (a vertex-exponential algorithm) of the graph. In this paper we present the first vertex-exponential algorithms to find a connected f-factor in a given graph. In fact, we give a vertex-exponential algorithm for the *editing* version of this problem, which is much more general than the problem of simply finding a connected f-factor. More formally, in this problem, which we call Min-Cost Edge Editing to f-factor (Min-Cost EFF), the input consists of a graph G, a function $f : V(G) \to [n]$, a cost function c on the edges and non-edges of G, and a target cost $c^\star$. The objective is to check if there is a sequence of non-edge additions and edge deletions with a total cost at most $c^\star$ such that the resulting graph is a connected f-factor. This problem generalizes the problem of finding a connected f-factor in a graph, even with the additional restriction that the edge costs are bounded polynomially in the size of $V(G)$.

Our main result is a *randomized* algorithm which, given an instance $(G, f, c, c^\star)$ of Min-Cost EFF where $c(\{v, w\})$ is bounded by a polynomial in $|V(G)|$ for every $v, w \in E(G)$, solves it in time $\mathcal{O}^*(2^n)$.

▶ **Theorem 1.** *There is a randomized algorithm that, given an instance $(G, c, c^\star)$ of* Min-Cost Edge Editing to f-factor *with the cost function c being bounded by a polynomial in $|V(G)|$, runs in time $\mathcal{O}^*(2^{|V(G)|})$ and either returns a solution or correctly (with high probability) concludes that one does not exist.*

We then extend this result to a "parity version" Connected parity f-factor of the problem where, given a graph G and a function f where where f is a mapping from $V(G)$ to $\{0, 1\}$, the objective is to check if G has a connected spanning subgraph H where the degree of every vertex v in the subgraph H is $f(v) \pmod 2$.

▶ **Theorem 2.** *There is a randomized algorithm that, given an instance (G, f) of* Connected parity f-factor *where $|V(G)| = n$, runs in time $\mathcal{O}^*(2^n)$ and either returns a solution or correctly (with high probability) concludes that one does not exist.*

This generalizes the problem of checking whether a graph has an *Eulerian* subgraph; this is a connected subgraph whose degrees are all even ($f(v) \equiv 0$). As our third major result we show that the *edge-weighted* versions of finding connected (parity) f-factors can also be solved in randomized $\mathcal{O}^*(2^n)$ time as long as the weights are bounded polynomially in n.

The main technical ingredients in our solutions for each of these problems have the same flavour: For each problem, we transform the input graph into an auxiliary graph in such a way that the connected solutions which we seek correspond, in a certain sense, to the set of *perfect matchings* of the auxiliary graph. Our algorithms rely on the notion of Tutte matrices of graphs and related algebraic techniques – introduced by Lovász [16] and recently used in [2, 6, 21, 11] – to phrase our problems in terms of looking for "non-zero" monomials of certain polynomials. To solve these latter problems we test whether the polynomials are identically zero over certain fields. The randomization in our algorithm arises from this final step of polynomial identity testing.

Related work. Moser and Thilikos [18] and Mathieson and Szeider [17] initiated the study of the parameterized complexity of editing a given graph to obtain a graph that satisfies certain specified degree constraints. Mathieson and Szeider in particular described an auxiliary graph where perfect matchings captured the editing solutions in the same flavor of Tutte's auxiliary graph capturing f-factors via perfect matchings. More recently Golovach has studied the parameterized complexity of editing to connected graphs under degree constraints [10]. Cai and Yang [3], Cygan et al. [5] and Fomin and Golovach [8] have all studied the parameterized complexity of deleting edges to obtain subgraphs with parity constraints on the degrees.

Organization of the rest of the paper. In Section 2 we describe our notation and some preliminary results. In Section 3 we take up the edge-editing version of our problem, MIN-COST EDGE EDITING TO f-FACTOR, and prove Theorem 1. In Section 4 we take up the parity version CONNECTED PARITY f-FACTOR and prove Theorem 2. We conclude in Section 5.

2 Preliminaries

We follow the graph notation and terminology of Diestel [7]. For a positive integer n we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We use A_{ij} to denote the element in the i^{th} row and j^{th} column of a matrix A. A subgraph H of a graph G is a *spanning subgraph* of G if $V(G) = V(H)$. We use $deg_G(v)$ to denote the degree of a vertex v in graph G and $N_G(v)$ to denote the neighbourhood of v in G; we omit the subscript when there is no scope for ambiguity. For a subset $S \subseteq V(G)$ of the vertex set of a graph G we use $G[S]$ to denote the subgraph *induced* by the set S and $G - S$ to denote the subgraph $G[V(G) \setminus S]$. For a subset S of vertices, we denote by $E(G)[S, V(G) \setminus S]$ the edges of G with an end point each in S and $V(G) \setminus S$. If F is a set of edges in a graph G, then we use $V(F)$ to denote the set of all vertices which form end-points of the edges in F. A *matching* in a graph G is any set M of edges in G such that no two edges of M have an end-point in common, and a matching M of G is a *perfect matching* if $V(M) = V(G)$. Let $w : E(G) \to \mathbb{Z}$ be function which assigns integer weights to the edges of a graph G. The *weight* of a subgraph H of G is then the sum $\sum_{e \in E(H)} w(e)$.

When we refer to expanded forms of *succinct* representations (such as summations and determinants) of polynomials, we use the term *naïve expansion* (or summation) to denote that expanded form of the polynomial which is obtained by merely writing out the operations indicated by the succinct representation. We use the term *simplified expansion* to denote the expanded form of the polynomial which results after we apply all possible simplifications (such as cancellations) to a naïve expansion. We call a monomial m which has a non-zero coefficient in a simplified expansion of a polynomial P, a *surviving* monomial of P in the simplified expansion.

▶ **Definition 3.** (Tutte matrix) The *Tutte matrix* of a graph G with n vertices is an $n \times n$ skew-symmetric matrix T over the set $\{x_{ij} | 1 \leq i < j \leq |V(G)|\}$ of indeterminates whose $(i,j)^{th}$ element is defined to be

$$T(i,j) = \begin{cases} x_{ij} & \text{if } \{i,j\} \in E(G) \text{ and } i < j \\ -x_{ji} & \text{if } \{i,j\} \in E(G) \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

We use $\mathcal{T}(G)$ to denote the Tutte matrix of graph G. We say that the variable x_{ij} is the *label* of edge $\{i,j\} \in E(G)$.

The following basic facts about the Tutte matrix $\mathcal{T}(G)$ of a graph G are well-known. When evaluated over any field of characteristic two, the determinant and the permanent of the matrix $\mathcal{T}(G)$ (indeed, of any matrix) coincide:

$$\det \mathcal{T}(G) = \text{perm}(\mathcal{T}(G)) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} \mathcal{T}(G)(i, \sigma(i)), \tag{1}$$

where S_n is the set of all *permutations* of $[n]$. Moreover, there is a one-to-one correspondence between the set of all *perfect matchings* of the graph G and the *surviving monomials* in the above expression for $\det \mathcal{T}(G)$ when its simplified expansion is computed over any field of characteristic two:

▶ **Proposition 1.** [16] *If $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_\ell, j_\ell)\}$ is a perfect matching of a graph G, then the product $\prod_{(i_k, j_k) \in M} x_{i_k j_k}$ appears as a surviving monomial in the sum on the right-hand side of Equation 1 when this sum is expanded and simplified over any field of characteristic two. Conversely, each surviving monomial in a simplified expansion of this sum over a field of characteristic two is of the form $\prod_{(i_k, j_k) \in M} x_{i_k j_k}$ where $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_\ell, j_\ell)\}$ is a perfect matching of G. In particular, $\det \mathcal{T}(G)$ is identically zero when expanded and simplified over a field of characteristic two if and only if graph G does not have a perfect matching.*

▶ **Lemma 4.** *(Schwartz-Zippel)*[19, 22] *Let $P(x_1, \ldots, x_n)$ be a multivariate polynomial of degree at most d over a field $\mathbb{F}$ such that P is not identically zero. Furthermore, let $r_1, \ldots, r_n$ be chosen uniformly at random from $\mathbb{F}$. Then, $Prob[P(r_1, \ldots, r_n) = 0] \leq \frac{d}{|\mathbb{F}|}$.*

We also require the following well-known interpolation lemma.

▶ **Lemma 5.** *Let $P(x)$ be a univariate polynomial of degree r over a field of size at least $r+1$. Then, given $r+1$ evaluations of $P(x)$, the polynomial can be determined in time polynomial in r.*

3 Editing to f-factors

The problem we study in this section is Edge Editing to f-factor. The formal definition of this problem is as follows.

Edge Editing to f-factor

Input:	Graph $G = (V, E)$, function $f : V \to \mathbb{N}$, k.
Question:	Can G be converted to a connected f-factor with at most k edge deletions and additions?

A set of k edge additions and deletions is referred to as a k-editing. For a given graph G, if $G - S_1 + S_2$ is an f-factor where S_1 is a set of edges of G and S_2 is a set of non-edges, then we refer to (S_1, S_2) as an ℓ-editing of G to an f-factor, where $\ell = |S_1 \cup S_2|$. It is easy to show that Editing to Connected f-factor where $f = 2$ is a generalization of Hamiltonicity (see for example [10]).

We begin with the following observation which relates local editing of subgraphs of G to f-factors on the one hand and the global editing of G to an f-factor on the other. We then subsequently define an auxiliary graph where perfect matchings capture editings to f-factors (see [17]).

▶ **Observation 1.** *Let G be a graph, let $f : V \to \mathbb{N}$, and let $S \subseteq V(G)$. Suppose the subgraphs $G[S]$ and $G - S$ have ℓ_1 and ℓ_2-editing to f-factors (S, F_1) and $((V(G) \setminus S), F_2)$ respectively and let ℓ_3 be the number of edges in the set $E(G)[S, V(G) \setminus S]$. Then, the union of the two editings along with the deletion of the edges in $E(G)[S, V(G) \setminus S]$ is an $(\ell_1 + \ell_2 + \ell_3)$-editing to the* disconnected *f-factor $(V(G), F_1 \uplus F_2)$. Similarly, let (S_1, S_2) be an editing of G to an f-factor $H = (V(G), F)$ and C be the union of some connected components of H. Let $S_1' = S_1 \cap \binom{V(C)}{2}$ and $S_2' = S_2 \cap \binom{V(C)}{2}$. Then, (S_1', S_2') is an editing to the f-factor C of the induced subgraph $G[V(C)]$.*

▶ **Definition 6** (Editing f-Blowup)**.** Let G be a graph and let $f : V(G) \to \mathbb{N}$ be such that $f(v) \leq deg(v)$ for each $v \in V(G)$. Let H be a graph and w be a weight function on the edges of H defined as follows

1. For each vertex v of G, we add a vertex set $A(v)$ of size $f(v)$ to H.
2. For each edge $e = \{v, w\}$ of G we add to H vertices v_e and w_e and edges (u, v_e) for every $u \in A(v)$ and (w_e, u) for every $u \in A(w)$. We assign weight 0 to all these edges. Finally, we add the edge (v_e, w_e) and set $w(v_e, w_e) = 2$.
3. For each non-edge $\bar{e} = \{v, w\}$ of G we add to H vertices $v_{\bar{e}}$ and $w_{\bar{e}}$ and edges $(u, v_{\bar{e}})$ for every $u \in A(v)$ and $(w_{\bar{e}}, u)$ for every $u \in A(w)$. We assign weight 1 to each of these edges. Finally, we add the edge $(v_{\bar{e}}, w_{\bar{e}})$ and set $w(v_{\bar{e}}, w_{\bar{e}}) = 0$.

This completes the construction. The graph H along with the weight function $w : E(H) \to \{0, 1, 2\}$ is called the editing *f-blowup* of graph G. We use $\mathcal{E}_f(G)$ to denote the editing f-blowup of G. We omit the subscript when there is no scope for ambiguity.

▶ **Definition 7** (Induced Editing f-blowup)**.** For a subset $S \subseteq V(G)$, we define the editing f-blowup of G *induced* by S as follows. Let the editing f-blowup of G be (H, w). Begin with the graph H and for every edge $e = (v, w) \in E(G)$ such that $v \in S$ and $w \notin S$, delete the vertices v_e and w_e. Similarly, for every non-edge $\bar{e} = (v, w) \notin E(G)$ such that $v \in S$ and $w \notin S$, delete the vertices $v_{\bar{e}}$ and $w_{\bar{e}}$. Let the graph H' be the union of those connected components of the resulting graph which contain the vertex sets $A(v)$ for vertices $v \in S$. Then, the pair (H', w) is called the editing f-blowup of G *induced* by the set S and is denoted by $\mathcal{E}_f(G)[S]$.

The construction of the editing f-blowup of G can be informally described as taking the complete graph on $V(G)$, making $f(v)$ "equivalent copies" of every vertex $v \in V(G)$, replacing every edge and non-edge of G by a path of length 3, and assigning weight 2 to the "middle" edge of the paths corresponding to an edge of G, assigning weight 1 to the "end" edges of the path corresponding to a non-edge of G and weight 0 to all other edges. Similarly, the construction of the editing f-blowup of G induced by a subset $S \subseteq V(G)$ can be described analogously starting with the graph $G[S]$.

We now prove a lemma (see also [17]) which gives an equivalence between editings to f-factors and perfect matchings in the editing f-blowup.

▶ **Lemma 8.** *A graph G has an ℓ-editing to an f-factor with $\ell \leq k$ if and only if the editing f-blowup of G, (H, w), has a perfect matching of weight at most $2k$.*

Proof. Let (S_x, S_y) be an editing to an f-factor $(V(G), F)$ of G such that $|S_x \cup S_y| \leq k$, where $F = (E(G) \setminus S_x) \cup S_y$. We now define the following matching M in H. For every pair $(v, w) \in \binom{V}{2} \setminus F$, if $e = (v, w) \in E(G)$ then we add the edge (v_e, w_e) to M and if $\bar{e} = (v, w) \notin E(G)$ then we add the edge $(v_{\bar{e}}, w_{\bar{e}})$ to M. For every edge $(v, w) \in F$, if $e = (v, w) \in E(G)$ then we add the edges (u, v_e) and (u', w_e) to M where u and u' are two vertices in $A(v)$ and $A(w)$ respectively such that they are as yet unsaturated by M. Similarly, for every edge $(v, w) \in F$, if $\bar{e} = (v, w) \notin E(G)$ then we add the edges $(u, v_{\bar{e}})$ and $(u', w_{\bar{e}})$ to M where u and u' are two vertices in $A(v)$ and $A(w)$ respectively such that they are as yet unsaturated by M. Since $|A(v)| = f(v)$ for every $v \in V(G)$, M indeed saturates the sets $A(v)$ for every $v \in V(G)$ and therefore is a perfect matching. We now consider the weight of M. Clearly, $E(G) \setminus F = S_x$ and the weight contributed to M by the edges of H corresponding these edges is $2|S_x|$. Similarly, the weight contributed to M by the edges of H corresponding to those in $S_y = F \setminus E(G)$ is $2|S_y|$. Therefore, $w(M) \leq 2k$. This completes the proof of the forward direction.

Conversely, suppose that H has a perfect matching M of weight at most $2k$. Let $S_x = \{e = (v, w) | (v, w) \in E(G) \wedge (v_e, w_e) \in M\}$ and $S_y = \{\bar{e} = (v, w) | (v, w) \notin E(G) \wedge (v_{\bar{e}}, w_{\bar{e}}) \notin M\}$. Observe that for every $\bar{e} = (v, w) \in S_y$, there is a vertex $u \in A(v)$ and $u' \in A(w)$ such that $(u, v_{\bar{e}})$,$(u', w_{\bar{e}}) \in M$. This is because the vertex $v_{\bar{e}}$ ($w_{\bar{e}}$) has exactly one neighbor disjoint from $A(v)$ (respectively $A(w)$) and by assumption, $(v_{\bar{e}}, w_{\bar{e}}) \notin M$. Since each edge of the form $(u, v_{\bar{e}})$ (where $u \in A(v)$) has weight 1 and occurs in M along with an edge $(u', w_{\bar{e}})$ of weight 1 (with $\bar{e} = (v, w)$), we conclude that $2|S_x \cup S_y| = w(M) \leq 2k$. We now claim that $(V(G), F)$ is an f-factor, where $F = (E(G) \setminus S_x) \cup S_y$. Let M_A be all those edges of M incident on $\bigcup_{v \in V(G)} A(v)$. Starting from H, we define a subgraph H' of G as follows. For each $v \in V(G)$, we identify all vertices $A(v)$ in H. We then contract every edge in M_A. It is easy to see that the resulting graph is indeed an f-factor of G. Furthermore, by definition, the edges in M_A are precisely those corresponding to the edges in F. This completes the proof of the lemma. ◀

Having established the relation between perfect matchings in the f-blowup and editings to f-factors, we now recall the definition of a "weighted" Tutte matrix (see for example [11]) which allows us to handle edge weights as this will be crucially required to encode the size of the editings.

▶ **Definition 9.** (Weighted Tutte matrix) The *Weighted Tutte matrix* of a graph G with n vertices and a weight function $w : E(G) \to \mathbb{Z}$ is an $n \times n$ skew-symmetric matrix T over the set $\{x_{ij} | 1 \leq i < j \leq |V(G)|\} \cup \{z\}$ of indeterminates whose $(i, j)^{th}$ element is defined to be

$$T(i,j) = \begin{cases} x_{ij} z^{w(i,j)} & \text{if } (i,j) \in E(G) \text{ and } i < j \\ -x_{ji} z^{w(i,j)} & \text{if } (i,j) \in E(G) \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

We use $\mathcal{T}_z(G)$ to denote the Weighted Tutte matrix of graph G.

The following proposition is analogous to Proposition 1 and the proof is identical.

▶ **Proposition 2.** *If $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_\ell, j_\ell)\}$ is a perfect matching of a graph G with a weight function w on its edges, then the product $(\prod_{(i_k, j_k) \in M} x_{i_k j_k}) \cdot z^{\Sigma_{(i_k, j_k) \in M} w(i_k, j_k)}$ appears as a surviving monomial in the sum on the right-hand side of Equation 1 when* *applied to $\mathcal{T}_z(G)$ (instead of $\mathcal{T}(G)$) and the sum is expanded and simplified over any field of characteristic two. Conversely, each surviving monomial in a simplified expansion of this sum over a field of characteristic two is of the form $(\prod_{(i_k,j_k)\in M} x_{i_k j_k}) \cdot z^{\sum_{(i_k,j_k)\in M} w(i_k,j_k)}$ where $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_\ell, j_\ell)\}$ is a perfect matching of G. In particular, $\det \mathcal{T}_z(G)$ is identically zero when expanded and simplified over a field of characteristic two if and only if graph G does not have a perfect matching.*

▶ **Definition 10.** With every set $S \subseteq V(G)$, we associate a specific monomial m_S which is defined to be the product taken over all $e = (v, w) \in E(G)[S, V(G) \setminus S]$ of the terms $x_{ij} z^{w(i,j)}$ where $\{i, j\} = \{v_e, w_e\}$ and over all $\bar{e} = (v, w) \in E(\bar{G})[S, V(G) \setminus S]$ of the terms $x_{ij} z^{w(i,j)}$ where $\{i, j\} = \{v_{\bar{e}}, w_{\bar{e}}\}$, where the terms $v_e, w_e, v_{\bar{e}}, w_{\bar{e}}$ are as in Definition 6 of the editing f-blowup $\mathcal{E}(G)$ of G. If $S = V(G)$, then we set $m_S = 1$.

In the spirit of [6], we now fix an arbitrary vertex $v^\star$ of G and define a polynomial $P(\bar{x}, z)$ over the indeterminates from the weighted Tutte matrix $\mathcal{T}_z(\mathcal{E}(G))$ of the f-blowup of G, as follows:

$$P(\bar{x}, z) = \sum_{S \subseteq V(G)\,;\, v^\star \in S} (\det \mathcal{T}_z(\mathcal{E}(G)[S])) \cdot (\det \mathcal{T}_z(\mathcal{E}(G)[V(G) \setminus S])) \cdot m_S, \tag{2}$$

where if a graph H has no vertices or edges then we set $\det \mathcal{T}(H) = 1$. In the sequel we use $\mathcal{F}$ to denote an arbitrary field of characteristic two. Observe that $P(\bar{x}, z)$ can be rewritten as $\sum_{i=0}^{r} Q_i(\bar{x}) \cdot z^i$ where r is an upper bound on the degree of z in any term of the polynomial $P(\bar{x}, z)$. We refer to the polynomial $Q_i(\bar{x})$ as the *coefficient* of z^i in $P(\bar{x}, z)$. Furthermore, every monomial m in the naïve expansion of $Q_i(\bar{x})$ is also referred to as a coefficient of z^i.

▶ **Definition 11.** We say that an editing (S_1, S_2) of G to an f-factor $(V(G), F)$ *contributes* a monomial $x_{i_1 j_1} \ldots x_{i_r j_r}$ to the naïve expansion of the right-hand side of Equation 2 if and only if the following conditions hold.

- For every $e = (v, w) \in F \cap E(G)$ ($\bar{e} = (v, w) \in F \setminus E(G)$), there is a $u \in A(v)$, $u' \in A(w)$ and $1 \le p, q \le r$ such that $\{u, v_e\} = \{i_p, j_p\}$ and $\{u', w_e\} = \{i_q, j_q\}$ (respectively $\{u, v_{\bar{e}}\} = \{i_p, j_p\}$ and $\{u', w_{\bar{e}}\} = \{i_q, j_q\}$).
- For every $e = (v, w) \in E(G) \setminus F$ ($\bar{e} = (v, w) \notin E(G) \cap F$), there is a $1 \le p \le r$ such that $\{v_e, w_e\} = \{i_p, j_p\}$ (respectively $\{v_{\bar{e}}, w_{\bar{e}}\} = \{i_p, j_p\}$).
- For every $1 \le p, q \le r$, if $\{u, v_e\} = \{i_p, j_p\}$ and $\{u', w_e\} = \{i_q, j_q\}$ for some $e \in F \cap E(G)$(respectively $\{u, v_{\bar{e}}\} = \{i_p, j_p\}$ and $\{u', w_{\bar{e}}\} = \{i_q, j_q\}$ for some $\bar{e} \in F \setminus E(G)$), then e (respectively $\bar{e}$) is in F.
- For every $1 \le p \le r$, if $\{i_p, j_p\} = \{v_e, w_e\}$ for some $e \in E(G)$ ($\{i_p, j_p\} = \{v_{\bar{e}}, w_{\bar{e}}\}$ for some $\bar{e} \notin E(G)$), then e (respectively $\bar{e}$) is not in F.
- For every $S \subseteq V(G)$ containing $v^\star$, such that S is a union of the vertex sets of (some) connected components of $(V(G), F)$, there is a pair of monomials m_1 and m_2 such that m_1 is a surviving monomial in the simplified expansion of $\det \mathcal{T}(\mathcal{E}(G)[S])$, m_2 is a surviving monomial in the simplified expansion of $\det \mathcal{T}(\mathcal{E}(G)[V(G) \setminus S])$, and $m_1 \cdot m_2 \cdot m_S = x_{i_1 j_1} \ldots x_{i_r j_r} \cdot z^{\sum_{k=1}^{r} w(i_k, j_k)}$.

Having set up the required notation, we now state the main lemma which allows us to show that monomials contributed by "undesirable editings" do not survive in the simplified expansion of the right hand side of Equation 2.

▶ **Lemma 12.** *Let G be a graph and (S_1, S_2) be an ℓ-editing of G to an f-factor $(V(G), F)$. Then,*

1. *All monomials contributed by (S_1, S_2) are coefficients of $z^{2\ell}$ in the naïve expansion of the right-hand side of Equation 2.*
2. *If $(V(G), F)$ is a* disconnected *f-factor of G then every monomial contributed by (S_1, S_2) occurs an* even *number of times in the polynomial $Q_{2\ell}(\bar{x})$ in the naïve expansion of the right-hand side of Equation 2.*
3. *If $(V(G), F)$ is a* connected *f-factor of G, then every monomial contributed by (S_1, S_2) occurs exactly once in the polynomial $Q_{2\ell}(\bar{x})$ in the naïve expansion of the right-hand side of Equation 2.*

As a consequence of the above lemma, we prove the following.

▶ **Lemma 13.** *The coefficient of $z^{2\ell}$ in the naïve expansion of $P(\bar{x}, z)$ is not identically zero over $\mathcal{F}$ if and only if G has an ℓ-editing to a connected f-factor.*

Proof. Observe that as a consequence of Proposition 2 combined with the proof of Lemma 8, we have that each surviving monomial in the naïve expansion of the right-hand side of Equation 2 is contributed by some editing to an f-factor of the graph G.

By this observation, every monomial which is a coefficient of $z^{2\ell}$ is contributed by an ℓ-editing to an f-factor and by Lemma 12, we have that every monomial contributed by this editing occurs an even number of times if and only if the resulting f-factor is disconnected. This completes the proof of the lemma. ◀

We now prove the main result of this section by giving an algorithm for editing to connected f-factors.

▶ **Theorem 14.** *There is a randomized algorithm that, given an instance (G, k) of* Editing to f-factor*, runs in time $\mathcal{O}^*(2^{|V(G)|})$ and either returns a solution or correctly (with high probability) concludes that one does not exist.*

Proof. Observe that the total degree of the polynomial $P(\bar{x}, z)$ is bounded by $n^2 + 2\binom{n}{2} + 2\binom{n}{2} \leq 3n^2$, where the sum of the first two terms is an upper bound on the number of vertices in the editing f-blowup which gives a bound on the degree of a monomial in $P(\bar{x}, z)$ due to $\bar{x}$ and the third term is a bound on the degree of a monomial due to z. We select values for the variables in $\bar{x}$ uniformly at random from a field $\mathcal{F}$ of characteristic 2 and size at least $3n^d$ for some fixed $d \geq 5$. Having fixed this instantiation of the variables in $\bar{x}$, we select $r = 2\binom{n}{2} + 1$ values for z from the field $\mathcal{F}$ and evaluate the polynomial $P(\bar{x}, z)$ for each of these r instantiations and return Yes if and only if for some $\ell \leq 2k$, the coefficient of z^ℓ is non-zero in the univariate polynomial $R(z)$ obtained by evaluating $P(\bar{x}, z)$ at the randomly selected points for $\bar{x}$. The r evaluations of the polynomial can be done in time $\mathcal{O}^*(2^n)$ by determinant computation and testing for a z^ℓ with non-zero coefficient in $R(z)$ can be done in polynomial time by interpolation (Lemma 5). This proves the stated bound on the running time of the algorithm. Therefore, it only remains to prove the correctness of the algorithm.

Suppose that (S_1, S_2) is a p-editing to a connected f-factor for some $p \leq k$. Then, by Lemma 13, we have that the coefficient of z^{2p}, $Q_{2p}(\bar{x})$, is not identically zero over $\mathcal{F}$. By the Schwartz-Zippel Lemma, we have that since $Q_{2p}(\bar{x})$ is not identically zero, then with probability at least $1 - \frac{1}{n^3}$ the evaluation of $Q_{2p}(\bar{x})$ at the randomly chosen points results in a non-zero value, implying that the coefficient of z^{2p} is non-zero in the polynomial $R(z)$. By the union bound, the probability that the coefficient of z^ℓ is "erroneously" zero in $R(z)$ for every $1 \leq \ell \leq 2k$ is at most $\frac{2k}{n^3} \leq \frac{1}{n}$. Therefore, if G has a p-editing to a connected f-factor with $p \leq k$, then with probability at least $1 - \frac{1}{n}$, we will detect the presence of such an editing. This completes the proof of the theorem. ◀

Finally, we note that if we are also given costs on the edges of the graph that are bounded polynomially in n, then we can also solve the version of the problem where costs are placed on the editing operations, in the same asymptotic running time with the only change appearing in the choice of the field from which $\bar{x}$ is instantiated at random. More precisely, we have the following theorem.

▶ **Theorem 1.** *There is a randomized algorithm that, given an instance $(G, c, c^\star)$ of* Min-Cost Edge Editing to f-factor *with the cost function c being bounded by a polynomial in $V(G)$, runs in time $\mathcal{O}^*(2^{|V(G)|})$ and either returns a solution or correctly (with high probability) concludes that one does not exist.*

The problem of finding a connected f-factor in a given graph is special case of Min-Cost Edge Editing to f-factor and hence we have the following corollary.

▶ **Corollary 15.** *There is a randomized algorithm that, given an instance (G, f) of* Connected f-factor *where $|V(G)| = n$, runs in time $\mathcal{O}^*(2^n)$ and either returns a solution or correctly (with high probability) concludes that one does not exist.*

4 Parity f-factors

In this section we extend our approach to handle the parity version. Most of the proof is identical to the arguments in the previous section, and so we focus on defining the new kind of f-blowup which we need, and a description of the corresponding matching characterization.

▶ **Definition 16.** Given a graph G and a function $f : V(G) \to \{0, 1\}$, a *parity f-factor* of graph G is a spanning subgraph H of G in which every vertex v has degree exactly $f(v) \pmod 2$. A *connected parity f-factor* of G is such a *connected* subgraph H of G.

▶ **Definition 17** (Parity f-Blowup)**.** Let G be a graph and let $f : V(G) \to \{0, 1\}$. Let H be a graph defined as follows

1. For each vertex v of G, we add a vertex set $A(v)$ which has size $deg(v)$ if $deg(v) \equiv f(v) \pmod 2$ and size $deg(v) - 1$ otherwise.
2. For each edge $e = \{v, w\}$ of G we add vertices v_e and w_e and edges (u, v_e) for every $u \in A(v)$ and (w_e, u) for every $u \in A(w)$. Finally, we add the edge (v_e, w_e).
3. For each v such that $f(v) = 0$, we choose an arbitrary pair of vertices a_v and a'_v in $A(v)$ and make a clique on the rest of the vertices of $A(v)$. For each v such that $f(v) = 1$, we choose an arbitrary vertex a_v in $A(v)$ and make a clique on the rest of the vertices of $A(v)$.

This completes the construction. The graph H is called the *parity f-blowup* of graph G. We use $\mathcal{P}_f(G)$ to denote the parity f-blowup of G . We omit the subscript when there is no scope for ambiguity.

▶ **Definition 18** (Induced Parity f-blowup)**.** For a subset $S \subseteq V(G)$, we define the parity f-blowup of G *induced* by S as follows. Let the parity f-blowup of G be H. Begin with the graph H and for every edge $e = (v, w) \in E(G)$ such that $v \in S$ and $w \notin S$, delete the vertices v_e and w_e. Let the union of connected components of the resulting graph containing the vertices of the set S be the graph H'. Then, the graph H' is called the parity f-blowup of G *induced* by the set S and is denoted by $\mathcal{P}_f(G)[S]$.

▶ **Lemma 19.** *A graph G has a parity f-factor if and only if the parity f-blowup of G has a perfect matching.*

Proof. Suppose that G has a parity f-factor $(V(G), F)$. We now define a matching M in the parity f-blowup of G as follows. For every $e \in E(G) \setminus F$, we add the edge (v_e, w_e) to M. For every edge $(v, w) \in F$, we add the edges (u, v_e) and (u', w_e) to M where u and u' are two vertices in $A(v)$ and $A(w)$ respectively such that they are as yet unsaturated by M. However, if either of a_v or a'_v is unsaturated at this point, we chose to saturate one of these and similarly for a_w and a'_w.

Since $|A(v)| \equiv f(v) \pmod 2$ and $|A(v)| \geq deg(v) - 1$ for every $v \in V(G)$, we conclude that M saturates $B(v)$ vertices from the set $A(v)$ for every $v \in V(G)$, where $B(v) \equiv f(v) \pmod 2$. Furthermore, since $(V(G), F)$ is a parity f-factor, $\{a_v, a'_v\} \subseteq B(v)$ for every v. The only unsaturated vertices in H at this point are the vertices in $A(v) \setminus B(v)$ for every $v \in V(G)$. However, since $B(v) \equiv f(v) \pmod 2$, we have that $B(v) \equiv |A(v)| \pmod 2$, implying that $|A(v) \setminus B(v)| \equiv 0 \pmod 2$. Since $\{a_v, a'_v\} \subseteq B(v)$ for every v, the subgraph $H[A(v) \setminus B(v)]$ is an even-sized clique and therefore we pick an arbitrary perfect matching in this clique and add it to M to get a perfect matching.

Conversely, suppose that M is a perfect matching of H. We define the set F as follows. For every $e = (v, w) \in E(G)$ such that $(v_e, w_e) \notin M$, we add the edge (v, w) to F. It can be argued along similar lines as before that $(V(G), F)$ is indeed a parity f-factor of G. This completes the proof of the lemma. ◀

Given the above definition of f-blowups and the structural lemma "equating" parity f-factors to perfect matchings in the f-blowup, the proof of the following theorem is identical to the proof of Theorem 14.

▶ **Theorem 2.** *There is a randomized algorithm that, given an instance (G, f) of* Connected Parity f-Factor *where $|V(G)| = n$, runs in time $\mathcal{O}^*(2^n)$ and either returns a solution or correctly (with high probability) concludes that one does not exist.*

▶ **Corollary 20.** *There is a randomized algorithm that, given a graph G, $|V(G)| = n$, runs in time $\mathcal{O}^*(2^n)$ and either returns a connected Eulerian subgraph of G with the maximum (or minimum) number of edges, or correctly (with high probability) concludes that one does not exist.*

5 Conclusion

In this paper we studied certain generalizations of the well-studied NP-hard problems Hamiltonicity and Max/Min-Eulerian Subgraph. We gave $\mathcal{O}^*(2^n)$ time randomized algorithms for the problems of finding connected f-factors in a graph, minimum editing to obtain a connected f-factor and finding a connected parity f-factor. The most natural direction forward in this line of research would be towards obtaining a *deterministic* vertex exponential algorithm as well as algorithms that handle super-polynomial weights.

References

1 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the Association of Computing Machinery*, 9(1):61–63, 1962.

2 Andreas Björklund. Determinant sums for undirected hamiltonicity. In *FOCS*, pages 173–182, 2010.

3 Leizhen Cai and Boting Yang. Parameterized complexity of even/odd subgraph problems. *J. Discrete Algorithms*, 9(3):231–240, 2011.

4 Gérard Cornuéjols. General factors of graphs. *J. Comb. Theory, Ser. B*, 45(2):185–198, 1988.

5 Marek Cygan, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Ildikó Schlotter. Parameterized complexity of eulerian deletion problems. *Algorithmica*, 68(1):41–61, 2014.

6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159, 2011.

7 Reinhard Diestel. *Graph Theory.* Springer-Verlag, Heidelberg, 3rd edition, 2005.

8 Fedor V. Fomin and Petr A. Golovach. Parameterized complexity of connected even/odd subgraph problems. *J. Comput. Syst. Sci.*, 80(1):157–179, 2014.

9 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

10 Petr A. Golovach. Editing to a connected graph of given degrees. *CoRR*, abs/1308.1802, 2013.

11 Gregory Gutin, Magnus Wahlström, and Anders Yeo. Parameterized rural postman and conjoining bipartite matching problems. *CoRR*, abs/1308.2599, 2013.

12 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

13 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

14 László Lovász. The factorization of graphs. *Combinatorial Structures and Their Applications*, pages 243–246, 1970.

15 László Lovász. The factorization of graphs. ii. *Acta Mathematica Academiae Scientiarum Hungarica*, 23(1–2):223–246, 1972.

16 László Lovász. On determinants, matchings, and random algorithms. In L. Budach, editor, *Fundamentals of Computation Theory FCT'79*, pages 565–574, Berlin, 1979. Akademie-Verlag.

17 Luke Mathieson and Stefan Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *J. Comput. Syst. Sci.*, 78(1):179–191, 2012.

18 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *J. Discrete Algorithms*, pages 181–190, 2009.

19 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

20 William Thomas Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of Mathematics*, 6:347–352, 1954.

21 Magnus Wahlström. Abusing the tutte matrix: An algebraic instance compression for the k-set-cycle problem. In *STACS*, pages 341–352, 2013.

22 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation*, volume 72, pages 216–226, 1979.

Connecting Vertices by Independent Trees*

Manu Basavaraju[1], Fedor V. Fomin[1], Petr A. Golovach[1], and Saket Saurabh[1,2]

1 **Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway**
`{manu.basavaraju,fedor.fomin,petr.golovach}@ii.uib.no`

2 **Institute of Mathematical Sciences, Chennai, India**
`saket@imsc.res.in`

Abstract

We study the paramereteized complexity of the following connectivity problem. For a vertex subset U of a graph G, trees $T_1, \dots, T_s$ of G are completely independent spanning trees of U if each of them contains U, and for every two distinct vertices $u, v \in U$, the paths from u to v in $T_1, \dots, T_s$ are pairwise vertex disjoint except for end-vertices u and v. Then for a given $s \geq 2$ and a parameter k, the task is to decide if a given n-vertex graph G contains a set U of size at least k such that there are s completely independent spanning trees of U. The problem is known to be NP-complete already for $s = 2$. We prove the following results:

- For $s = 2$ the problem is solvable in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.
- For $s = 2$ the problem does not admit a polynomial kernel unless NP $\subseteq$ coNP /poly.
- For arbitrary s, we show that the problem is solvable in time $f(s,k)n^{\mathcal{O}(1)}$ for some function f of s and k only.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Parameterized complexity, FPT-algorithms, completely independent spanning trees

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.73

1 Introduction

Two spanning trees T_1 and T_2 of a graph G are *independent* if they are rooted in the same vertex r and for every vertex $v \neq r$ of G, the two (v, r)-paths, one in T_1 and one in T_2, are internally disjoint, i. e. having no edge and no internal vertex in common. Independent spanning trees have applications to fault-tolerant protocols in distributed processor networks [3, 11]. In 2001, Hasunuma in [7, 8] introduced the notion of *completely independent spanning trees*, an interesting variant of the classical notion of connectivity. Formally, spanning trees $T_1, \dots, T_s$ of a graph G are *completely independent* if for every two distinct vertices $u, v \in V(G)$, the (u, v)-paths in $T_1, \dots, T_s$, are pairwise vertex disjoint except for end-vertices u and v.

The problem of deciding whether a graph G has two completely independent spanning trees is NP-complete [8]. Since not every graph has even two completely independent spanning trees, the following optimization version of the problem is meaningful. For a given

* Supported by the European Research Council (ERC) via grant Rigorous Theory of Preprocessing, reference 267959.

$s \geq 2$, can one find a maximum set of vertices spanned by s completely independent trees? More precisely, for a set of vertices U of a graph G, we say that a subgraph T of G is a *spanning tree of* U if T is an inclusion-minimal tree in G containing all vertices of U. Spanning trees $T_1, \dots, T_s$ of U are completely independent if for any two distinct vertices $u, v \in U$, the (u, v)-paths in $T_1, \dots, T_s$, are pairwise vertex disjoint except for end-vertices u and v. Then the task is to find a set of vertices U of maximum size (we call the vertices of U *terminals*) such that there are s completely independent spanning trees of U.

In this paper, we initiate the study of the following parameterized problem.

Independently s-Connected k-Set
Instance: A graph G and positive integers $s \geq 2$ and k.
Parameter 1: s.
Parameter 2: k.
Question: Does G contain a set of terminals U of size at least k such that there are s completely independent spanning trees of U?

Previous results. Hasunama [8] has shown that it is NP-complete to decide whether a graph G has two completely independent spanning trees. He also obtained a number of results about existence of completely independent spanning trees for some special graph classes. Other, mostly combinatorial, studies of the problem were carried out by Hasunuma and Morisaka [9] and Péterfalvi [12].

Our contribution. Our main result is stated in the following theorem.

▶ **Theorem 1.** Independently 2-Connected k-Set *can be solved in time* $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ *for* n*-vertex graphs.*

We prove the theorem by applying a WIN/WIN approach. We start with a combinatorial result, which is interesting on its own. In Section 3 we show that every 2-connected graph of pathwidth at least k, contains as a minor a graph H, which is a tree on k vertices plus one vertex adjacent to all other vertices. We also give a polynomial time algorithm which either provide us H, or a path decomposition of width $k - 1$. As it is sufficient to solve Independently 2-Connected k-Set for the blocks of the input graph, we either obtain two completely independent spanning trees for k terminals, or construct a path decomposition of width at most $k - 1$. The next step is an algorithm given in Section 5 that solves Independently 2-Connected k-Set in time single exponential in the treewidth of the input graph. This step is based on the recent techniques of computing representative sets of graphic matroids [4]. Combining together both cases, we obtain the proof of Theorem 1.

Let us remark, that the NP-hardness reduction in [8] from Not-All-Equal-3SAT reduces to a graph of size linear in the number of variables and clauses of the formula. Thus, unless the Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [10] fails, there is no $2^{o(k)}n^{\mathcal{O}(1)}$ algorithm for Independently 2-Connected k-Set and thus our upper bound is asymptotically tight up to ETH.

We complement our algorithm with a complexity result on kernelization for Independently 2-Connected k-Set, namely that the problem does not admit a polynomial kernel unless NP $\subseteq$ coNP /poly.

We also show that Independently s-Connected k-Set is FPT when parameterized by $s + k$. It is not hard to reduce Independently s-Connected k-Set to the problem of finding a topological minor of constant size in a graph. Then the result follows from a deep Theorem of Grohe, Kawarabayashi, Marx and Wollan [6] on the parameterized testing of topological minors.

2 Preliminaries

Graphs. We consider finite undirected graphs without loops or multiple edges. The vertex set of a graph G is denoted by $V(G)$ and the edge set is denoted by $E(G)$. For a set of vertices $S \subseteq V(G)$, $G[S]$ denotes the subgraph of G induced by S, and by $G - S$ we denote the graph obtained from G by the removal of all the vertices of S, i. e., the subgraph of G induced by $V(G) \setminus S$. For a single element set $\{v\}$, we write $G - v$ instead of $G - \{v\}$. For a vertex v, we denote by $N_G(v)$ its *(open) neighborhood* in G, that is, the set of vertices which are adjacent to v. The *degree* of a vertex v is denoted by $d_G(v) = |N_G(v)|$, and $\Delta(G)$ is the maximum degree of G. A vertex v is a *cut-vertex* of G if $G - v$ has more connected components than G. A connected graph with at least two vertices is 2-*connected* if it does not contain a cut-vertex. A maximal 2-connected subgraph of G is called a 2-*connected component* or *block* of G. Let T be a tree. For a vertex $v \in V(T)$, we say that v is a *leaf* if $d_T(v) = 1$ or $d_T(v) = 0$ (if $|V(T)| = 1$), and we say that v is an *internal* vertex otherwise.

Minors. The *edge contraction* of $e = uv$ removes u and v from G, and replaces them by a new vertex adjacent to precisely those vertices to which u or v were adjacent. If u is a vertex of degree two such that its neighbors x, y are not adjacent, then the *vertex dissolution* of u removes u and adds a new edge xy. A graph H is a *minor* of G if H can be obtained from a subgraph of G by a sequence of vertex deletions, edge deletions and edge contractions. Alternatively, we can define minors as follows. For two non-empty vertex disjoint subsets $X_1, X_2 \subseteq V(G)$, X_1 and X_2 are *adjacent* if there is $uv \in E(G)$ such that $u \in X_1$ and $v \in X_2$. An *H-witness structure* $\mathcal{W}$ is a collection of $|V(H)|$ non-empty vertex disjoint subsets $W(x) \subseteq V(G)$, one for each $x \in V(H)$, called *H-witness sets*, such that each $W(x)$ induces a connected subgraph of G, and for all $x, y \in V(H)$ with $x \neq y$, if x and y are adjacent in H, then $W(x)$ and $W(y)$ are adjacent in G. It is straightforward to see that H is a minor of G if and only if G has an H-witness structure. A graph H is a *topological minor* of G if H can be obtained from a subgraph of G by a sequence of vertex deletions, edge deletions and vertex dissolution. Notice that if H is a topological minor of G, then by subdividing edges of H we can obtain a graph that is isomorphic to a subgraph of G.

Treewidth and pathwidth. A *tree decomposition* of a graph G is a pair (X, T) where T is a tree and $X = \{X_i \mid i \in V(T)\}$ is a collection of subsets (called *bags*) of $V(G)$ such that:
1. $\bigcup_{i \in V(T)} X_i = V(G)$,
2. for each edge $xy \in E(G)$, $x, y \in X_i$ for some $i \in V(T)$, and
3. for each $x \in V(G)$, the set $\{i \mid x \in X_i\}$ induces a connected subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\}, T)$ is $\max_{i \in V(T)} \{|X_i| - 1\}$. The *treewidth* of a graph G (denoted as $\mathbf{tw}(G)$) is the minimum width over all tree decompositions of G.

If T is restricted to be a path, then (X, T) is said to be a *path decomposition*. Respectively, the *pathwidth* of a graph G (denoted as $\mathbf{pw}(G)$) is the minimum width over all path decompositions of G. Whenever we consider a path decomposition (X, P), we assume that the bags are enumerated in the path order with respect to P. In other words, a path decomposition of G is a sequence of bags $(X_1, \ldots, X_r)$.

3 Algorithm for Independently 2-Connected k-Set

In this section we design an algorithm for INDEPENDENTLY 2-CONNECTED k-SET. We start by a simple characterization of completely independent spanning trees that we use in our arguments. This is followed by a a structural result that shows that if the pathwidth of the input graph is large then the given instance is a YES instance. We use this to design a algorithm mentioned in Theorem 1.

3.1 Characterization of completely independent spanning trees

Hasunuma proved in [7] that if $T_1, \ldots, T_s$ are spanning trees of a graph G, then $T_1, \ldots, T_s$ are completely independent if and only if $T_1, \ldots, T_s$ are edge-disjoint and for any vertex $v \in V(G)$, there is at most one spanning tree T_i such that $d_{T_i}(v) > 1$. We need a similar claim for completely independent spanning trees of a set of terminals.

▶ **Lemma 2.** *Let G be a graph, and let $U \subseteq V(G)$ with $|U| = k$. Let also $T_1, \ldots, T_s$ be spanning trees of U. Then $T_1, \ldots, T_s$ are completely independent spanning trees of U if and only if*

1. *$T_1, \ldots, T_s$ are edge disjoint,*
2. *for all $i, j \in \{1, \ldots, s\}$, $i \neq j$, if $v \in V(T_i) \cap V(T_j)$, then $v \in U$,*
3. *for each $v \in U$, there is at most one $i \in \{1, \ldots, s\}$ such that $d_{T_i}(v) > 1$.*

Proof. We assume that $k, s \geq 2$, as the claim is trivial otherwise. We first show the forward direction. Suppose that $T_1, \ldots, T_s$ are completely independent spanning trees of U.

We show that for any $i, j \in \{1, \ldots, s\}$, $i \neq j$, T_i and T_j have no common vertex that is an internal vertex of both the trees. To obtain a contradiction, assume that $u \in V(T_i) \cap V(T_j)$ is an internal vertex of both T_i and T_j. The vertex u is a cut-vertex of T_i. Because T_i is an inclusion-minimal tree that contains U, there are two terminals $x, y \in U$ that are in two distinct components T_i' and T_i'' of $T_i - u$ respectively. The tree T_j has the unique (x, y)-path P and $u \notin V(P)$. Since u is an internal vertex of T_j, $T_j - u$ has at least two components, and P lies completely in one component T_j' of $T_j - u$. By minimality, there is $z \in U$ such that z is in another component of $T_j - u$. Notice that $z \notin V(T_i')$ or $z \notin V(T_i'')$. Assume without loss of generality that $z \notin V(T_i')$. Because $x \in V(P)$ and z are in distinct components of $T_j - u$, u is an internal vertex of the (x, z)-path in T_j. Because $z \notin V(T_i')$ and $x \in V(T_i')$, u is an internal vertex of the (x, z)-path in T_i as well, but it contradicts the assumption that $T_1, \ldots, T_s$ are completely independent spanning trees of U.

The proved claim immediately implies (3). To show (1), assume that two distinct trees T_i, T_j have a common edge uv. Because neither u nor v can be an internal vertex of the both trees, we can assume without loss of generality that u is a leaf of T_i and v is a leaf of T_j. Because T_i, T_j are inclusion-minimal trees that contains U, any leaf of T_i or T_j is a terminal, and $u, v \in U$. Then we have that the (u, v)-paths in T_i and T_j have a common edge; a contradiction. To prove (2), it is sufficient to observe that if $v \in V(T_i) \cap V(T_j)$ and $v \notin U$, then by minimality of T_i, T_j, v is an internal vertex of both these trees, a contradiction.

Assume now that $T_1, \ldots, T_s$ are spanning trees of U that satisfy (1)–(3). Consider any distinct $u, v \in U$ and $i, j \in \{1, \ldots, s\}$. Let P_i, P_j be the (u, v)-paths in T_i and T_j respectively. By (1), P_i and P_j are edge disjoint. If P_i and P_j have a common vertex $x \neq u, v$, then by (2), $x \in U$, and then $d_{T_i}(x), d_{T_j}(x) \geq 2$ contradicting (3). Hence, P_i and P_j are internally vertex disjoint. ◀

Clearly, if G is a disconnected subgraph, then G has a set of terminals U of size at least k such that there are s completely independent spanning trees of U if and only if there is such a set of terminals in one of the components of G, i. e., we can consider only connected graphs. Lemma 2 implies that we can restrict ourself by 2-connected graphs. To see it, it is sufficient to observe that if a set of terminals U has two vertices that does not belong to the same block, then there is a cut-vertex of G that is an internal vertex of any spanning tree of U contradicting Lemma 2.

▶ **Lemma 3.** *Let G be a connected graph. For positive integers s and k, G has a set of terminals U of size at least k such that there are s completely independent spanning trees of U in G if and only if there is a block H of G with the same property.*

3.2 Independent trees and pathwidth

In this section we show that if a 2-connected graph G has pathwidth at least k, then G has a set of terminals U of size at least k such that there are two completely independent spanning trees of U. We need some additional notations. Let G be a graph. For $Z \subseteq V(G)$, $\mathbf{att}(Z)$ is the set of all $v \in Z$ with a neighbor in $V(G) \setminus Z$, and $\alpha(Z) = |\mathbf{att}(Z)|$.

▶ **Theorem 4.** *Let G be a 2-connected graph with n vertices and m edges. Let also k be a positive integer. If $\mathbf{pw}(G) \geq k$, then G has a minor H with the property that there is a vertex $w \in V(H)$ such that $d_H(w) \geq k$ and $H - w$ is a tree. Moreover, there is an algorithm that in time $\mathcal{O}(nm)$ either produces a witness structure of such a minor H, or constructs a path decomposition of G of width at most $k - 1$.*

Proof. Suppose that Z is a non-empty proper subset of $V(G)$ that satisfies the following conditions:

(i) $1 \leq \alpha(Z) \leq k$,

(ii) there are vertex disjoint connected subgraph $C_0, \ldots, C_t$ of $G[Z]$ where $t = \alpha(Z) - 1$ such that
- for each $i \in \{0, \ldots, t\}$, $V(C_i) \cap \mathbf{att}(Z) \neq \emptyset$,
- G has an edge with one end-vertex in C_0 and another in C_i for all $i \in \{1, \ldots, t\}$, and
- $V(C_1) \cup \ldots \cup V(C_t)$ are in the same component of $G - V(C_0)$.

(iii) $G[Z]$ has a path decomposition $(X_1, \ldots, X_r)$ of width at most $k-1$ such that $\mathbf{att}(Z) \subseteq X_r$.

Notice that $\mathbf{att}(Z) \subseteq V(C_0) \cup \ldots \cup V(C_t)$ and each C_i has the unique vertex in $\mathbf{att}(Z)$.

We prove the following claim.

▶ **Claim A.** *Either $\alpha(Z) = k$ and G has a minor H with the property that there is a vertex $w \in V(H)$ such that $d_H(w) \geq k$ and $H - w$ is a tree, or $|V(G) \setminus Z| = 1$ and $\mathbf{pw}(G) \leq k - 1$, or there is Z' such that $Z \subset Z' \subset V(G)$ and Z' satisfies (i)–(iii).*

Proof of Claim A. Suppose that $\alpha(Z) = k = t + 1$. Consider $u \in \mathbf{att}(Z) \cap V(C_0)$. There is a neighbor v of u in $V(G) \setminus Z$. Let C_{t+1} be the subgraph of G with the unique vertex v. The graph G is 2-connected. Then $G - u$ is connected, and G has a path that joins v with at least one of $C_1, \ldots, C_t$ that avoids C_0. Because $V(C_1) \cup \ldots \cup V(C_t)$ are in the same component of $G - V(C_0)$, we have that $V(C_1) \cup \ldots \cup V(C_{t+1})$ also are in the same component of $G - V(C_0)$. Now we construct the minor H of G as follows. We contract the edges of C_0 and denote the obtained vertex w. Then we contract the edges of the subgraphs $C_1, \ldots, C_k$ and denote the obtained vertices by $u_1, \ldots, u_k$ respectively. Let G' be the obtained graph. The vertices $u_1, \ldots, u_k$ are in the same component of $G' - w$. Hence, $G' - w$ has a tree T that contains $u_1, \ldots, u_k$. We remove the vertices of $V(G') \setminus (V(T) \cup \{w\})$. Finally, we remove all the edges of the obtained graph except the edges of T and the edges that join w and T. Because $u_1, \ldots, u_k \in V(T)$ are adjacent to w, we have a required minor.

Let now $\alpha(Z) < k$ and let $|V(G) \setminus Z| = 1$. By (iii), $G[Z]$ has a path decomposition $(X_1, \ldots, X_r)$ of width at most $k-1$ such that $\mathbf{att}(Z) \subseteq X_r$. Let $X_{r+1} = \mathbf{att}(Z) \cup (V(G) \setminus Z)$. It is straightforward to see that $(X_1, \ldots, X_{r+1})$ is a path decomposition of G of width at most $k-1$.

From now we assume that $\alpha(Z) < k$ and $|V(G) \setminus Z| > 1$. We show that the set Z can be extended by one vertex in such a way that the obtained set satisfies (i)–(iii). Let $u \in \mathbf{att}(Z) \cap V(C_0)$ and let v be an arbitrary neighbor of u in $V(G) \setminus Z$. We set $Z' = Z \cup \{v\}$ and let $X_{r+1} = \mathbf{att}(Z) \cup \{v\}$.

Because $V(G) \setminus Z' \neq \emptyset$ and G is connected, $\alpha(Z') \geq 1$. Clearly, $\alpha(Z') \leq \alpha(Z) + 1 \leq k$. Hence, (i) holds.

It is straightforward to verify that $(X_1, \ldots, X_{r+1})$ is a path decomposition of $G[Z']$ and $\mathbf{att}(Z') \subseteq \mathbf{att}(Z) \cup \{v\} \subseteq X_{r+1}$. The width of this decomposition is $\max\{w, t+1\}$ where w is the width of $(X_1, \ldots, X_r)$. Recall that $w \leq k-1$ and $t + 1 = \alpha(Z) < k$. It means that (iii) is fulfilled.

It remains to show (ii). Let C_{t+1} be the subgraph of G with the unique vertex v. Clearly, $\mathbf{att}(Z') \subseteq V(C_0) \cup \ldots \cup V(C_{t+1})$ and G has an edge with one end-vertex in C_0 and another in C_i for all $i \in \{1, \ldots, t+1\}$. Since G is 2-connected, $G - u$ is connected, and G has a path that joins v with at least one of $C_1, \ldots, C_t$ that avoids C_0. Because $V(C_1) \cup \ldots \cup V(C_t)$ are in the same component of $G - V(C_0)$, we have that $V(C_1) \cup \ldots \cup V(C_{t+1})$ also are in the same component of $G - V(C_0)$. Notice that it can happen that not all C_i have vertices in $\mathbf{att}(Z')$. Let $\{C'_1, \ldots, C'_{t'}\} = \{C_i | V(C_i) \cap \mathbf{att}(Z') \neq \emptyset, 1 \leq i \leq t+1\}$. Because $V(C_1) \cup \ldots \cup V(C_{t+1})$ are in the same component of $G - V(C_0)$, $V(C'_1) \cup \ldots \cup V(C'_{t'})$ are in the same component of $G - V(C_0)$ too. Observe that since $|V(C_i) \cap \mathbf{att}(Z)| = 1$ for $i \in \{0, 1, \ldots, t\}$, we have $|V(C'_i) \cap \mathbf{att}(Z')| = 1$ for $i \in \{1, \ldots, t'\}$, $|V(C_0) \cap \mathbf{att}(Z')| \leq 1$, and $\mathbf{att}(Z') \subseteq V(C_0) \cup V(C'_1) \ldots \cup V(C'_{t'})$. We consider two cases.

Case 1. The vertex u has at least two neighbors in $V(G) \setminus Z$. Then C_0 has the unique vertex u in $\mathbf{att}(Z')$, and we have that $\alpha(Z') = t' + 1$ and (ii) holds for $C_0, C'_1, \ldots, C'_{t'}$.

Case 2. The vertex v is the unique neighbor of u in $V(G) \setminus Z$. Observe that since G is 2-connected, $t' \geq 2$ in this case. Consider the graph G' obtained from G by contracting edges of $C'_1, \ldots, C'_{t'}$ and denote by $x_1, \ldots, x_{t'}$ the vertices obtained from these graphs respectively. We have that $x_1, \ldots, x_{t'}$ are in the same component of $G' - V(C_0)$. We construct a spanning tree T for $\{x_1, \ldots, x_{t'}\}$ in $G' - V(C_0)$. Because $t' \geq 2$, T has at least two leaves. Without loss of generality we assume that x_1 is a leaf of T. Then $x_2, \ldots, x_{t'}$ and, consequently, $V(C'_2), \ldots, V(C'_{t'})$ are in the same component of $G' - (V(C_0) \cup \{x_1\})$ and $G - (V(C_0) \cup V(C'_1))$ respectively. We construct C'_0 by taking $C_0 \cup C'_1$ and adding an edge that joins C_0 and C'_1. Then $\mathbf{att}(Z') \subseteq V(C'_0) \cup V(C'_2) \ldots \cup V(C'_{t'})$ and G has an edge with one end-vertex in C'_0 and another in C'_i for all $i \in \{2, \ldots, t'\}$. Also $V(C'_2) \cup \ldots \cup V(C'_{t'})$ are in the same component of $G - V(C'_0)$. Because $V(C'_1) \cap \mathbf{att}(Z') \neq \emptyset$, $|V(C'_0) \cap \mathbf{att}(Z')| = 1$. Then $\alpha(Z') = t'$ and (ii) is fulfilled for $C'_0, C'_2, \ldots, C'_{t'}$. ◀

Observe that a non-empty proper subset Z of $V(G)$ that satisfies (i)–(iii) always exists, because for any vertex $z \in V(G)$, $Z = \{z\}$ satisfies (i)–(iii). Suppose that $\mathbf{pw}(G) \geq k$, and let $Z \subset V(G)$ be an inclusion-maximal non-empty proper subset of $V(G)$ that satisfies (i)–(iii). Then by Claim A, G has a minor H with the property that there is a vertex $w \in V(H)$ such that $d_H(w) \geq k$ and $H - w$ is a tree.

To complete the proof, it remains to observe that the proof of Claim A can be transformed to an algorithm that either constructs H, or produces a tree decomposition of G of width at most $k-1$, or increases Z by adding one vertex. In the last case the algorithm also modifies the subgraphs $C_0, \ldots, C_t$ and adds a new bag to the path decomposition. Initially we choose an arbitrary vertex z and set $Z = \{z\}$, $t = 0$ and C_0 has the unique vertex z. Since each iteration can be done in time $\mathcal{O}(m)$ and we have at most n iterations, we conclude that the algorithm runs in time $\mathcal{O}(nm)$. ◀

This combinatorial result is tight in the following sense. If $G = K_k$, then $\mathbf{pw}(G) = k-1$, and G has a minor H with the property that there is a vertex $w \in V(H)$ such that $d_H(w) \geq k$ and $H-w$ is a tree. But clearly G has no minors with a vertex of degree at least k. Theorem 4 gives us the following corollary.

▶ **Corollary 5.** *Let G be a 2-connected graph with n vertices and m edges. Let also k be a positive integer. If $\mathbf{pw}(G) \geq k$, then G has a set of terminals U of size at least k such that there are 2 completely independent spanning trees of U. Moreover, there is an algorithm that in time $\mathcal{O}(nm)$ either produces U and completely independent spanning trees T_1, T_2 of U, or constructs a path decomposition of G of width at most $k-1$.*

We conclude this section by the observation that the bounds obtained in Corollary 5 is almost tight. If $G = K_k$ with $k \geq 4$, we have $\mathbf{pw}(G) = k-1$, and there are two completely independent spanning trees of $V(G)$ where $|V(G)| = k+1$ and the number of terminals cannot be increased.

3.3 Proof of Theorem 1

In this section we give a proof of Theorem 1 by combining Lemma 3 and Corollary 5. However, we also need the following lemma which gives an algorithm for Independently 2-Connected k-Set on graphs of bounded treewidth.

▶ **Lemma 6.** *Let G be an n-vertex graph given together with its tree decomposition of width* $\mathbf{tw}$*. Then* Independently 2-Connected *k-*Set *on G can be solved in time $2^{\mathcal{O}(\mathbf{tw})}n^{\mathcal{O}(1)}$.*

A naive algorithm for Independently 2-Connected k-Set would run in time $\mathbf{tw}^{\mathcal{O}(\mathbf{tw})}n^{\mathcal{O}(1)}$. To obtain the desired running time, we use the idea of representative families introduced in [4] in our dynamic programming algorithm. By Lemma 2, we know that for Independently 2-Connected k-Set we need to find two edge disjoint trees (F_1, F_2) satisfying certain properties. Thus, if we take the intersection of the solution to some subgraph of the input graph we get two forests (F_1', F_2'). Let G be the input graph and H be an induced subgraph of G such that $|\partial(H)| \leq t$ where $\partial(H) = N(V(G) \setminus V(H))$. We call H, a t-boundaried graph. At every node of the tree decomposition one can associate a $t+1$ boundaried graph H of G. For H, we keep a family of partial solutions $\mathcal{P}$ that satisfies a following property. Given a solution (L_1, L_2) to Independently 2-Connected k-Set, there is a partial solution $(Q_1, Q_2) \in \mathcal{P}$ such that $(Q_1 \cup L_1^r, Q_2 \cup L_2^r)$ is also a solution. Here, $L_1^r = L_1 \setminus E(H)$ and $L_2^r = L_2 \setminus E(H)$. We use the ideas of matroids and representative families in order to bound the size of $\mathcal{P}$. One views each of the partial solution, (Q_1, Q_2), as a pair of forests in a graphic matroid of a clique on the vertex set $\partial(H)$. Thus these forests correspond to a pair of independent sets in graphic matroid. Furthermore, for every solution (L_1, L_2) to Independently 2-Connected k-Set, we view (L_1^r, L_2^r) as another pair of independent sets in graphic matroid of a clique on the vertex set $\partial(H)$. Now one observes that $(Q_1 \cup L_1^r, Q_2 \cup L_2^r)$ forms a pair of spanning tree of some induced subgraph of the clique. Once we have identified partial solutions as pairs of independent sets in a matroid one can show that the size of $\mathcal{P}$ is upper bounded by $2^{\mathcal{O}(t)}$. We finally give the proof of our main result.

Proof of Theorem 1. Let (G, k) be an input to INDEPENDENTLY 2-CONNECTED k-SET. Also assume that G has n vertices and m edges. We first compute all the blocks of G, say $B_1, \ldots, B_\ell$, in $\mathcal{O}(m+n)$ time. Now, by Lemma 3 we know that G is a YES-instance if and only if there exists an $i \in \{1, \ldots, \ell\}$ such that (B_i, k) is a YES-instance. Now on each B_i, we first apply Corollary 5 and in $\mathcal{O}(nm)$ time either produce a terminal set U and completely independent spanning trees T_1, T_2 of U, or construct a path decomposition of B_i of width at most $k-1$. In the former case we return U and completely independent spanning trees T_1, T_2 of U. In the later case we apply Lemma 6 and check whether (G, k) is a YES-instance to INDEPENDENTLY 2-CONNECTED k-SET. This completes the proof. ◀

4 Lower Bound on Kernelization

We proved that INDEPENDENTLY 2-CONNECTED k-SET is FPT. Hence, it is natural to ask whether this problem has a polynomial kernel. A parameterized problem Π is said to admit a kernel of size $f\colon \mathbb{N} \to \mathbb{N}$ if every instance (x, k) can be reduced in polynomial time to an equivalent instance with both size and parameter value bounded by $f(k)$. When $f(k) = k^{\mathcal{O}(1)}$ then we say that Π *admits a polynomial* kernel. The study of kernelization has recently been one of the main areas of research in parameterized complexity, yielding many important new contributions to the theory. The development of a framework for ruling out polynomial kernels under certain complexity-theoretic assumptions [1, 2, 5] has added a new dimension to the field and strengthened its connections to classical complexity.

Using the results by Bodlaender et al. [1], we show that it is unlikely even if we restrict ourself to 2-connected graph. We first give a few definitions required for our proof. A *composition algorithm* for a parameterized problem Π is an algorithm that receives as an input a sequence of instances $(I_1, k), \ldots, (I_t, k)$ of Π where each I_i is an input and k is a parameter, and in time polynomial in $\sum_{i=1}^{t} |I_i| + k$ produces an instance (I', k') of Π such that i) (I', k') is a YES-instance of Π if and only if (I_i, k) is a YES-instance for some $i \in \{1, \ldots, t\}$, and ii) k' is polynomial in k. If Π has a composition algorithm, then it is said that Π is *compositional.* Bodlaender et al. [1] proved the following theorem.

▶ **Theorem 7** ([1]). *If Π is a compositional parameterized problem such that the unparameterized version of Π is* NP*-complete, then Π has no polynomial kernel unless* NP $\subseteq$ coNP /poly.

It is easy to see that INDEPENDENTLY 2-CONNECTED k-SET is compositional for general (or connected) graphs. But by Lemma 3, it is sufficient to consider the problem for 2-connected graphs. Hence, we prove the following theorem.

▶ **Theorem 8.** INDEPENDENTLY 2-CONNECTED k-SET *has no polynomial kernel even for 2-connected graphs unless* NP $\subseteq$ coNP /poly.

Proof. As the unparameterized version of INDEPENDENTLY 2-CONNECTED k-SET is NP-complete for 2-connected graphs by the results of Hasunuma in [8], it is sufficient to show that INDEPENDENTLY 2-CONNECTED k-SET is compositional for 2-connected graphs.

Let $(G_1, k), \ldots, (G_t, k)$ be a sequence of instances of INDEPENDENTLY 2-CONNECTED k-SET where $G_1, \ldots, G_t$ are 2-connected, and we assume without loss of generality that $k \geq 3$. Let also $n_i = |V(G_i)| \geq 3$ for $i \in \{1, \ldots, t\}$, and denote by $v_1^i, \ldots, v_{n_i}^i$ the vertices of G_i for $i \in \{1, \ldots, t\}$. We construct G' as follows (see Fig. 1).

- For each $h \in \{1, \ldots, t\}$ and for each ordered pair (i, j) of distinct $i, j \in \{1, \ldots, n_h\}$, construct a copy $G_h^{(i,j)}$ of G_h; denote by $x_h^{(i,j)}$ and $y_h^{(i,j)}$ the vertices v_i^h and v_j^h of the copy $G_h^{(i,j)}$ of G_h respectively.

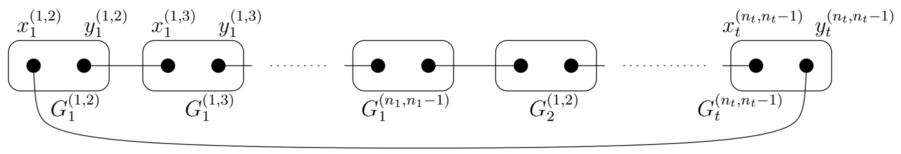


Figure 1 The construction of G'.

- For each $h \in \{1,\dots,t\}$, construct edges $y_h^{(i,j)}x_h^{(r,s)}$ for distinct ordered pairs $(i,j),(r,s)$ such that either $i=r$ and $s=j+1$ or $r=i+1$ and $j=n_h, s=1$.
- For each $h \in \{1,\dots,t\}$, construct edges $y_h^{(n_h,n_h-1)}x_{h+1}^{(1,2)}$; we assume here that $x_{t+1}^{(1,2)} = x_1^{(1,2)}$.

We let $k'=2k$. Notice that for all $x_h^{(i,j)}$ and $y_h^{(i,j)}$, G' has the unique edges that join these vertices with the vertices outside $G_h^{(i,j)}$. We call these edges by $x_h^{(i,j)}$ and $y_h^{(i,j)}$-*edges* respectively. Observe also that for all h,h' and $(i,j),(r,s)$, the graph G' has a $(y_h^{(i,j)},x_{h'}^{(r,s)})$-path that contains $y_h^{(i,j)}$ and $x_{h'}^{(r,s)}$-edges.

It is straightforward to see that G' is 2-connected. We show that (G',k') is a YES-instance of Independently 2-Connected k'-Set if and only if (G_h,k) is a YES-instance for some $h \in \{1,\dots,t\}$.

Suppose that there is $h \in \{1,\dots,t\}$ such that G_h has a set of terminals U of size at least k such that there are two completely independent spanning trees F,T of U. Because $k \geq 3$, F and T have internal vertices. We choose such vertices denoted by v_i^h are v_j^h respectively. By Lemma 2, $i \neq j$. Denote by $F_h^{(i,j)},T_h^{(i,j)}$ and $F_h^{(j,i)},T_h^{(j,i)}$ the copies of F,T in $G_h^{(i,j)}$ and $G_h^{(j,i)}$ respectively. Let P be a $(y_h^{(i,j)},x_h^{(j,i)})$-path in G' that contains $y_h^{(i,j)}$ and $x_h^{(j,i)}$-edges, and let Q be a $(y_h^{(j,i)},x_h^{(i,j)})$-path in G' that contains $y_h^{(j,i)}$ and $x_h^{(i,j)}$-edges. Let T' be the tree obtained by taking the union of $T_h^{(i,j)},T_h^{(j,i)}$ and P, and let F' be the tree obtained by taking the union of $F_h^{(i,j)},F_h^{(j,i)}$ and Q. It remains to observe that F',T' are completely independent spanning trees of U' where U' is the union of the copies of U in $G_h^{(i,j)}$ and $G_h^{(j,i)}$. Since $|U'|=2|U|\geq 2k$, we have that G a set of terminals U' of size at least k' such that there are two completely independent spanning trees F',T' of U'.

Suppose now that G a set of terminals U' of size at least k' such that there are two completely independent spanning trees F',T' of U'.

We claim that there are at most two $G_h^{(i,j)}$ that contain vertices of U'. To obtain a contradiction, assume that three distinct $G_{h_1}^{(i_1,j_1)},G_{h_2}^{(i_2,j_2)},G_{h_3}^{(i_3,j_3)}$ have vertices of U'. Then by the construction of G', there is $s \in \{1,2,3\}$ such that F' contains the $x_{h_s}^{(i_s,j_s)}$ and $y_{h_s}^{(i_s,j_s)}$-edges. Because F',T' are edge disjoint by Lemma 2, T' cannot contain any vertex of $G_{h_s}^{(i_s,j_s)}$; a contradiction. We consider two cases.

Case 1. The set U' contains vertices of the unique $G_h^{(i,j)}$. If F',T' do not include the $x_h^{(i,j)}$ and $y_h^{(i,j)}$-edges, then F',T' are subtrees of $G_h^{(i,j)}$. By taking the copies of F',T' in G_h, we have that G_h has a set of terminals of size at least $k'>k$ such that there are two completely independent spanning trees of the set. Suppose that one of the trees, say F', contains at least one of the $x_h^{(i,j)}$ and $y_h^{(i,j)}$-edges. Because F' is a minimal spanning tree of U', F' contains both the $x_h^{(i,j)},y_h^{(i,j)}$-edges. Then F' has the unique $(y_h^{(i,j)},x_h^{(i,j)})$-path P with these edges, and the internal vertices of P have degree two in F'. Then the forest obtained from F' by the deletion of the edges and the inner vertices of P has two components F_1 and F_2. Because $V(F')\cap U=(V(F_1)\cap U)\cup(V(F_2)\cap U)$ and $U_1=(V(F_1)\cap U),U_2=(V(F_2)\cap U)$

are disjoint, we can assume without loss of generality that $|U_1| \geq k$. Let F be the unique minimal spanning subtree of U_1 in F_1. Because F' contains the $x_h^{(i,j)}$ and $y_h^{(i,j)}$-edges, T' is a subgraph of $G_h^{(i,j)}$ by Lemma 2. Let T be be the unique minimal spanning subtree of U_1 in T'. We have that $G_h^{(i,j)}$ has the set of terminals U_1 of size at least k such that there are two completely independent spanning trees F, T of U_1. By taking the copies of F, T in G_h, we obtain that G_h has a set of terminals of size at least k such that there are two completely independent spanning trees of the set.

Case 2. The set U' contains vertices of two distinct $G_h^{(i,j)}, G_{h'}^{(r,s)}$. Let $U_1 = V(G_h^{(i,j)}) \cap U'$ and $U_2 = V(G_{h'}^{(r,s)}) \cap U'$. Because U_1, U_2 is a partition of U', we can assume without loss of generality that $|U_1| \geq k$. Notice that F', T' contain the $x_h^{(i,j)}, y_h^{(i,j)}, x_{h'}^{(r,s)}, y_{h'}^{(r,s)}$-edges, and the $x_h^{(i,j)}, y_{h'}^{(r,s)}$-edges (the $y_h^{(i,j)}, x_{h'}^{(r,s)}$-edges respectively) are in the same tree. We assume that F' contains the $x_h^{(i,j)}, y_{h'}^{(r,s)}$-edges and T' has the $y_h^{(i,j)}, x_{h'}^{(r,s)}$-edges. Then F' has the unique $(x_h^{(i,j)}, y_{h'}^{(r,s)})$-path Q and and T' has the unique $(y_h^{(i,j)}, x_{h'}^{(r,s)})$-path R, and the internal vertices of Q and R have degree two in F' and T' respectively. Then the forest obtained from F' by the deletion of the edges and the inner vertices of Q has exactly two components F_1, F_2, and it can be assumed that F_1 is a subgraph of $G_h^{(i,j)}$ and F_2 is a subgraph of $G_{h'}^{(r,s)}$. Notice that $U_1 \subseteq V(F_1)$, and let F be the unique spanning tree of U_1 in F_1. By the same arguments, the forest obtained from T' by the deletion of the edges and the inner vertices of R has exactly two components T_1, T_2, and it can be assumed that T_1 is a subgraph of $G_h^{(i,j)}$ and T_2 is a subgraph of $G_{h'}^{(r,s)}$. Again, $U_1 \subseteq V(F_1)$, and we consider the unique spanning tree T of U_1 in T_1. We have that $G_h^{(i,j)}$ has the set of terminals U_1 of size at least k such that there are two completely independent spanning trees F, T of U_1. By taking the copies of F, T in G_h, we obtain that G_h has a set of terminals of size at least k such that there are two completely independent spanning trees of the set.

In the both cases we have that there is $h \in \{1, \ldots, t\}$ such that (G_h, k) is a YES-instance of Independently 2-Connected k-Set, and it competes the proof. ◀

5 FPT algorithm for Independently s-Connected k-Set and a generalization

In this section we design an algorithm for Independently s-Connected k-Set. In fact, what we show is that this problem is is FPT when parameterized by $k + s$. We show that this problem can be reduced to checking existence of the bounded number of topological minors of bounded size. As the checking of existence of topological minors can be done in FPT-time by the recent results of Grohe et al. [6], we obtain the following theorem.

▶ **Theorem 9.** Independently s-Connected k-Set *is* FPT *when parameterized by* $s+k$.

Proof. If $k = 1$ or $s = 1$, then Independently s-Connected k-Set is trivial. If $k = 2$, then the problem can be solved in polynomial time by checking the existence of two vertices that can be joined by at least s internally vertex disjoint paths. Also if $s = 2$, then Independently s-Connected k-Set is FPT when parameterized by k by Theorem 1. Hence, we can assume that $s, k \geq 3$.

We prove the following two claims.

▶ **Claim B.** *If H is a topological minor of G such that (H, s, k) is a YES-instance of* Independently s-Connected k-Set*, then (G, s, k) is a YES-instance of* Independently s-Connected k-Set.

Proof of Claim B. Suppose that (H, s, k) is a YES-instance of INDEPENDENTLY s-CONNECTED k-SET for a topological minor H of G. Then there is a set of terminals $U \subseteq V(H)$ of size at least k and there are s completely independent spanning trees $T_1, \dots, T_s$ of U in H. Since H is a topological minor of G, G has a subgraph H' such that H' can be obtained from H by a sequence of edge subdivisions. Let $T'_1, \dots, T'_s$ be the trees obtained from $T_1, \dots, T_s$ by applying these edge subdivisions to the edges of these trees. Denote by U' the set of vertices of G that correspond to the vertices of U in H'. It remains to observe that $T'_1, \dots, T'_s$ are completely independent spanning trees of U' in G by Lemma 2, i. e., (G, s, k) is a YES-instance of INDEPENDENTLY s-CONNECTED k-SET. ◀

▶ **Claim C.** *If (G, s, k) is a YES-instance of* INDEPENDENTLY s-CONNECTED k-SET*, then G has a topological minor H with at most $sk + k - 2s$ vertices such that (H, s, k) is a YES-instance of* INDEPENDENTLY s-CONNECTED k-SET*.*

Proof of Claim C. Suppose that (G, s, k) is a YES-instance of INDEPENDENTLY s-CONNECTED k-SET. Then there is a set of terminals $U \subseteq V(G)$ of size exactly k and there are s completely independent spanning trees $T_1, \dots, T_s$ of U in G. Let H be a subgraph of G that is the union of $T_1, \dots, T_s$. Denote by H' the graph obtained from H by the recursive dissolutions of degree two vertices that have non-adjacent neighbors. Clearly, H' is a topological minor of G. Notice that because $s \geq 3$, the vertices of U are not dissolved, and we can dissolve only internal vertices of $T_1, \dots, T_s$. Let $T'_1, \dots, T'_s$ be the trees obtained from $T_1, \dots, T_s$ respectively by these dissolutions. Then $T'_1, \dots, T'_s$ are completely independent spanning trees of U in H' by Lemma 2, i. e., (H', s, k) is a YES-instance of INDEPENDENTLY s-CONNECTED k-SET.

To obtain the bound on the number of vertices of H', we show that for each T_i, all non-terminal internal vertices of degree two of T_i are dissolved. To obtain a contradiction, assume that at some step, we could not dissolve a vertex u of degree two. It can happen only if u has the neighbors x and y that are adjacent. Because T_i is a tree and the terminals are not dissolved, x and y are joined in some other tree T_j, i. e., $x, y \in V(T_i) \cap V(T_j)$. Moreover, x and y are joined in T_i, T_j by the unique (x, y)-paths P_i, P_j respectively such that the internal vertices of P_i, P_j have degree two in T_i, T_j respectively. By Lemma 2, $x, y \in U$. Because $k \geq 3$, each of x, y is an internal vertex of one of the trees $T_1, \dots, T_s$ by Lemma 2. Since $s \geq 3$, either x or y is an internal vertex of at least two trees; a contradiction.

Thus, each T'_i has no non-terminal vertices of degree one or two. Therefore, because $|U| = k$, T'_i has at most $k-2$ internal vertices. Then the total number of internal vertices of $T'_1, \dots, T'_s$ is at most $s(k-2)$, and the total number of vertices of H' is at most $s(k-2)+k$. ◀

Now we can solve INDEPENDENTLY s-CONNECTED k-SET as follows. We consider all $2^{\mathcal{O}(s^2k^2)}$ graphs H with at most $sk + k - 2s$ vertices. For each H, we solve INDEPENDENTLY s-CONNECTED k-SET using, e. g., brute force. If we obtain a YES-answer, then we check whether H is a topological minor of G by the algorithm of Grohe et al. [6]. If H is a topological minor of G, then (G, s, k) is a YES-instance of INDEPENDENTLY s-CONNECTED k-SET by Claim B. If we have a NO-answer for all H, then INDEPENDENTLY s-CONNECTED k-SET for (G, s, k) has a NO-answer by Claim C. ◀

A similar result can be obtained for the variant of the problem where a set of terminals is fixed. Formally, INDEPENDENT TREES FOR A SET OF TERMINALS ask for a graph G, positive integer s and a set U, whether there are s completely independent spanning trees of U in G. Using the same arguments as in the proof of Theorem 9, we can show the following.

▶ **Theorem 10.** Independent Trees for a Set of Terminals *is* FPT *when parameterized by* $s + |U|$.

6 Conclusions

In this paper we initiated parameterized complexity of a natural connectivity problem and designed several FPT algorithms for it. We conclude with several open questions.

- Is it possible to solve Independently s-Connected k-Set in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ for a fixed $s \geq 3$?
- What can be said about the approximability of Independently s-Connected k-Set? Is there a constant factor approximation algorithm for the problem for $s = 2$?
- We have shown that Independent Trees for a Set of Terminals is FPT when parameterized by $s + |U|$. Is it possible to obtain a more efficient algorithm for this problem? In particular, is it possible to solve the problem in single-exponential in $|U|$ for $s = 2$?

References

1 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

2 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In Leonard J. Schulman, editor, *Proc. of the 42nd ACM Symp. on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5–8 June 2010*, pages 251–260. ACM, 2010.

3 Danny Dolev, Joseph Y. Halpern, Barbara Simons, and H. Raymond Strong. A new look at fault-tolerant network routing. *Inf. Comput.*, 72(3):180–196, 1987.

4 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In Chandra Chekuri, editor, *Proc. of the 25th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014*, pages 142–151. SIAM, 2014.

5 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. In Cynthia Dwork, editor, *Proc. of the 40th Annual ACM Symp. on Theory of Computing, Victoria, BC, Canada, May 17–20, 2008*, pages 133–142. ACM, 2008.

6 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In Lance Fortnow and Salil P. Vadhan, editors, *Proc. of the 43rd ACM Symp. on Theory of Computing, STOC 2011, San Jose, CA, USA, 6–8 June 2011*, pages 479–488. ACM, 2011.

7 Toru Hasunuma. Completely independent spanning trees in the underlying graph of a line digraph. *Discrete Mathematics*, 234(1-3):149–157, 2001.

8 Toru Hasunuma. Completely independent spanning trees in maximal planar graphs. In Ludek Kucera, editor, *Graph-Theoretic Concepts in Computer Science, 28th International Workshop, WG 2002, Cesky Krumlov, Czech Republic, June 13–15, 2002, Revised Papers*, volume 2573 of *Lecture Notes in Computer Science*, pages 235–245. Springer, 2002.

9 Toru Hasunuma and Chie Morisaka. Completely independent spanning trees in torus networks. *Networks*, 60(1):59–69, 2012.

10 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

11 Alon Itai and Michael Rodeh. The multi-tree approach to reliability in distributed networks. *Inf. Comput.*, 79(1):43–59, 1988.

12 Ferenc Péterfalvi. Two counterexamples on completely independent spanning trees. *Discrete Math.*, 312(4):808–810, 2012.

Tree Deletion Set Has a Polynomial Kernel (but no OPT$^{\mathcal{O}(1)}$ Approximation)

Archontia C. Giannopoulou*[1], Daniel Lokshtanov†[1], Saket Saurabh‡[1,2], and Ondřej Suchý§[3]

1 Department of Informatics, University of Bergen, Norway
{archontia.giannopoulou|daniello}@ii.uib.no

2 The Institute of Mathematical Sciences, Chennai, India
saket@imsc.res.in

3 Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
ondrej.suchy@fit.cvut.cz

Abstract

In the Tree Deletion Set problem the input is a graph G together with an integer k. The objective is to determine whether there exists a set S of at most k vertices such that $G \setminus S$ is a tree. The problem is NP-complete and even NP-hard to approximate within any factor of OPT^c for any constant c. In this paper we give an $\mathcal{O}(k^5)$ size kernel for the Tree Deletion Set problem. An appealing feature of our kernelization algorithm is a new reduction rule, based on system of linear equations, that we use to handle the instances on which Tree Deletion Set is hard to approximate.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Tree Deletion Set, Feedback Vertex Set, Kernelization, Linear Equations

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.85

1 Introduction

In the Tree Deletion Set problem we are given as input an undirected graph G and integer k, and the task is to determine whether there exists a set $S \subseteq V(G)$ of size at most k such that $G \setminus S$ is a tree, that is, a connected acyclic graph. This problem was first mentioned by Yannakakis [25] and is related to the classical Feedback Vertex Set problem. Here input is a graph G and integer k and the goal is to decide whether there exists a set S on at most k vertices such that $G \setminus S$ is acyclic. The only difference between the two problems is that in Tree Deletion Set $G \setminus S$ is required to be connected, while in Feedback Vertex Set it is not. Both problems are known to be NP-complete [10, 25].

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959.

† The research was supported by the Bergen Research Foundation and the University of Bergen through project "BeHard".

‡ The research was supported by the European Research Council through Starting Grant 306992 "Parameterized Approximation".

§ The research was partially supported by the grant 14-13017P of the Czech Science Foundation.

Despite the apparent similarity between the two problems their computational complexities differ quite dramatically. Feedback Vertex Set admits a factor 2-approximation algorithm, while Tree Deletion Set is known to not admit any approximation algorithm with ratio $\mathcal{O}(n^{1-\epsilon})$ for any $\epsilon > 0$, unless $\mathtt{P} = \mathtt{NP}$ [1, 25]. With respect to parameterized algorithms, the two problems exhibit more similar behavior. Indeed, some of the techniques that yield fixed parameter tractable algorithms for Feedback Vertex Set [4, 5] can be adapted to also work for Tree Deletion Set [21].

It is also interesting to compare the behavior of the two problems with respect to polynomial time preprocessing procedures. Specifically, we consider the two problems in the realm of *kernelization*. We say that a parameterized graph problem admits a *kernel* of size $f(k)$ if there exists a polynomial time algorithm, called a *kernelization algorithm*, that given as input an instance (G, k) to the problem outputs an equivalent instance (G', k') with $k' \leq f(k)$ and $|V(G')| + |E(G')| \leq f(k)$. If the function f is a polynomial, we say that the problem admits a *polynomial kernel*. We refer to the surveys [11, 18] for an introduction to kernelization. For the Feedback Vertex Set problem, Burrage et al. [3] gave a kernel of size $\mathcal{O}(k^{11})$. Subsequently, Bodlaender [2] gave an improved kernel of size $\mathcal{O}(k^3)$ and finally Thomassé [22] gave a kernel of size $\mathcal{O}(k^2)$. On the other hand the existence of a polynomial kernel for Tree Deletion Set was open until this work. It seems difficult to directly adapt any of the known kernelization algorithms for Feedback Vertex Set to Tree Deletion Set. Indeed, Raman et al. [21] conjectured that Tree Deletion Set does not admit a polynomial kernel.

The main reason to conjecture that Tree Deletion Set does not admit a polynomial kernel stems from an apparent relation between kernelization and approximation algorithms (cf. [19, page 15]). Most problems that admit a polynomial kernel, also have approximation algorithms with approximation ratio polynomial in OPT (cf. [14, page 2]). Here OPT is the value of the optimum solution to the input instance. In fact many kernelization algorithms are already approximation algorithms with approximation ratio polynomial in OPT.

This relation between approximation and kernelization led to a conjecture [20, 8] that Vertex Cover does not admit a kernel with $(2 - \epsilon)k$ vertices for $\epsilon > 0$, as this probably would yield a factor $(2 - \epsilon)$ approximation for the problem thus violating the Unique Games Conjecture [13].

It is easy to show that an approximation algorithm for Tree Deletion Set with ratio OPT$^{\mathcal{O}(1)}$ would yield an approximation algorithm for the problem with ratio $\mathcal{O}(n^{1-\epsilon})$ thereby proving $\mathtt{P} = \mathtt{NP}$. In particular, suppose Tree Deletion Set had an OPT^c algorithm for some constant c. Since the algorithm will never output a set of size more than n, the approximation ratio of the algorithm is upper bounded by $\min(\text{OPT}^c, \frac{n}{\text{OPT}}) \leq n^{1-\frac{1}{c+1}}$. This rules out approximation algorithms for Tree Deletion Set with ratio OPT$^{\mathcal{O}(1)}$, and makes it very tempting to conjecture that Tree Deletion Set does not admit a polynomial kernel.

In this paper we show that Tree Deletion Set admits a kernel of size $\mathcal{O}(k^5)$. To the best of our knowledge this is among the few examples of problems that do admit a polynomial kernel, but do not admit any approximation algorithm with ratio OPT$^{\mathcal{O}(1)}$ under plausible complexity assumptions. The only other example we are aware of is a special case of the CSP studied by Kratsch and Wahlström [15].

Our Methods. The starting point of our kernel are known reduction rules for Feedback Vertex Set adapted to our setting. We also adapt the strategy to model some "pendant parts" of the graph by weight on vertices during the kernelization process to simplify the structure of the graph. By applying these graph theoretical reduction rules we can show that there is a polynomial time algorithm that given an instance (G, k) of Tree Deletion Set outputs an equivalent instance (G', k') and a partition of $V(G')$ into sets B, T, and I such that

1. $|B| = \mathcal{O}(k^2)$,
2. $|T| = \mathcal{O}(k^4)$,
3. I is an independent set, and
4. for every $v \in I$, $N_{G'}(v) \subseteq B$, and $N_{G'}(v)$ is a double clique.

Here a "double clique" means that for every pair x, y of vertices in $N_{G'}(v)$, there are two edges between them. Thus we will allow G' to be a multigraph, and consider a double edge between two vertices as a cycle. In order to obtain a polynomial kernel for Tree Deletion Set it is sufficient to reduce the set I to size polynomial in k.

For every vertex $v \in I$ and tree deletion set S we know that $|N_{G'}(v) \setminus S| \leq 1$, since otherwise $G' \setminus S$ would contain a double edge. Further, if $v \notin S$ then v has to be connected to the rest of $G' \setminus S$ and hence $|N_{G'}(v) \setminus S| = 1$, implying that v is a leaf in $G' \setminus S$. Therefore $G' \setminus (S \cup I)$ must be a tree. We can now reformulate the problem as follows.

For each vertex u in $G' \setminus I$ we have a variable x_u which is set to 0 if $u \in S$ and $x_u = 1$ if $u \notin S$. For each vertex $v \in I$ we have a linear equation $\sum_{u \in N(v)} x_u = 1$. The task is to determine whether it is possible to set the variables to 0 or 1 such that (a) the subgraph of G' induced by the vertices with variables set to 1 is a tree and (b) the number of variables set to 0 plus the number of unsatisfied linear equations is at most k.

At this point it looks difficult to reduce I by graph theoretic means, as performing operations on these vertices correspond to making changes in a system of linear equations. In order to reduce I we prove that there exists an algorithm that given a set $\mathcal{S}$ of linear equations on n variables and an integer k in time $\mathcal{O}(|\mathcal{S}|n^{\omega-1}k)$ outputs a set $\mathcal{S}' \subseteq \mathcal{S}$ of at most $(n+1)(k+1)$ linear equations such that any assignment of the variables that violates at most k linear equations of $\mathcal{S}'$ satisfies all the linear equations of $\mathcal{S} \setminus \mathcal{S}'$. To reduce I we simply apply this result and keep only the vertices of I that correspond to linear equations in $\mathcal{S}'$. We believe that our reduction rule for linear equations will find more applications in the future and, while not as involved, adds a little to the toolbox of algebraic reduction rules for kernelization (see, for example, [7, 6, 17, 16, 23]).

Due to space constraints, the proofs of lemmata marked with $\star$ are deferred to the full version of the paper.

2 Basic Notions

For every positive integer n we denote by $[n]$ the set $\{1, 2, \ldots, n\}$, $\mathbb{N}$ denotes the set of positive integers, and $\mathbb{R}$ denotes the real numbers.

For a graph $G = (V, E)$, we use $V(G)$ to denote its vertex set V and $E(G)$ to denote its edge set E. If $S \subseteq V(G)$ we denote by $G \setminus S$ the graph obtained from G after removing the vertices of S. In the case where $S = \{u\}$, we abuse notation and write $G \setminus u$ instead of $G \setminus \{u\}$. For $S \subseteq V(G)$, the *neighborhood* of S in G, $N_G(S)$, is the set $\{u \in V(G) \setminus S \mid \exists v \in S : \{u, v\} \in E(G)\}$. Again, in the case where $S = \{v\}$ we abuse notation and write $N_G(v)$ instead of $N_G(\{v\})$. The degree of vertex v denoted $\deg(v)$ is the number of edges incident to it, loops being counted twice. A graph is connected if there is a path between any pair of its vertices. A connected component in a graph G is a set of vertices H such that $G[H]$ is connected and H is maximal with this property. We use $\mathcal{C}(G)$ to denote the set of the connected components of G. Given a graph G and a set $S \subseteq G$, we say that S is a *feedback* *vertex set* of G if the graph $G \setminus S$ does not contain any cycles. In the case where $G \setminus S$ is connected we call S *tree deletion set* of G. Moreover, given a set $S \subseteq V(G)$, we say that S is a double clique of G if every pair of vertices in S is joined by a double edge.

Given two vectors x and y we denote by $\mathbf{d}_H(x, y)$ the Hamming distance of x and y, that is, $\mathbf{d}_H(x, y)$ is equal to the number of positions where the vectors differ. For every $k \in \mathbb{N}$ we denote by $\mathbf{0}^k$ the k-component vector $(0, 0, \ldots, 0)$. When k is implied from the context we abuse notation and denote $\mathbf{0}^k$ as $\mathbf{0}$.

For a rooted tree T and vertex set M in $V(T)$ the least common ancestor-closure (*LCA-closure*) **LCA-closure**(M) is obtained by the following process. Initially, set $M' = M$. Then, as long as there are vertices x and y in M' whose least common ancestor w is not in M', add w to M'. Finally, output M' as the LCA-closure of M.

▶ **Lemma 1** (Fomin et al. [9]). *Let T be a tree, $M \subseteq V(T)$, and $M' =$ **LCA-closure**(M). Then, $|M'| \leq 2|M|$ and for every connected component C of $T \setminus M'$, $|N_T(C)| \leq 2$.*

3 A polynomial kernel for Tree Deletion Set

In this section we prove a polynomial size kernel for a weighted variant of the Tree Deletion Set problem. More precisely the problem we will study is following.

Weighted Tree Deletion Set (wTDS)

Instance:	A graph G, a function $w : V(G) \to \mathbb{N}$, and a non-negative integer k.
Parameter:	k.
Question:	Does there exist a set $S \subseteq V(G)$ such that $\sum_{v \in S} w(v) \leq k$ and $G \setminus S$ is a tree?

3.1 Known Reduction Rules for wTDS

In this subsection we state some already known reduction rules for wTDS that are going to be needed during our proofs.

▶ **Reduction Rule 1** (Raman et al. [21]). *If the input graph is disconnected, then delete all vertices in connected components of weight less than $(\sum_{v \in V} w(v)) - k$ and decrease k by the weight of the deleted vertices.*

▶ **Observation 2** (Raman et al. [21]). *If $\left(\sum_{v \in V} w(v)\right) > 2k$, then after the exhaustive application of Reduction Rule 1 the graph has at most one connected component.*

▶ **Reduction Rule 2** (Raman et al. [21]). *If v is of degree 1 and u is its only neighbor, then delete v and increase the weight of u by the weight of v.*

▶ **Reduction Rule 3** (Raman et al. [21]). *If $v_0, v_1, \ldots, v_l, v_{l+1}$ is a path in the input graph, such that $l \geq 3$ and $\deg(v_i) = 2$ for every $i \in [l]$, then replace the vertices $v_1, \ldots, v_l$ by two vertices u_1 and u_2 with edges $\{v_0, u_1\}$, $\{u_1, u_2\}$, and $\{u_2, v_{l+1}\}$ and with $w(u_1) = \min\{w(v_i) \mid i \in [l]\}$ and $w(u_2) = \left(\sum_{i=1}^{l} w(v_i)\right) - w(u_1)$.*

Given a vertex x of G, an *x-flower of order k* is a set of k cycles pairwise intersecting exactly in x. If G has an x-flower of order $k + 1$, then x should be in every tree deletion set of weight at most k as otherwise we would need at least $k + 1$ vertices to hit all cycles passing through x. Thus the following reduction rule is safe.

▶ **Reduction Rule 4.** *Let (G, w, k) be an instance of* wTDS*. If G has an x-flower of order at least $k + 1$, then remove x and decrease the parameter k by the weight of x. The resulting instance is $(G \setminus \{x\}, w|_{V(G)\setminus\{x\}}, k - w(x))$.*

The following theorem allows us to apply Reduction Rule 4 exhaustively in polynomial time. A version of the theorem appears also in [2], but the version given in [22] is significantly more powerful.

▶ **Theorem 3** (Thomassé [22])**.** *Let G be a multigraph and x be a vertex of G without a self loop. Then in polynomial time we can find an x-flower of order $k + 1$ or, if such an x-flower does not exist, a set of vertices $Z \subseteq V(G) \setminus \{x\}$ of size at most $2k$ intersecting every cycle containing x.*

▶ **Reduction Rule 5.** *Let (G, w, k) be an instance of* wTDS*. If v is a vertex such that $w(v) > k + 1$, then let $w(v) = k + 1$.*

An instance (G, w, k) of wTDS is called *semi-reduced* if none of the Reduction Rules 1–5 can be applied. By Observation 2 such an instance is either connected or the total weight of all vertices is at most $2k$ and hence we have a kernel. Therefore, for the rest of the paper we assume that the instance is connected.

▶ **Lemma 4** ($\star$)**.** *If (G, w, k) is an instance of* wTDS *reduced with respect to Reduction Rule 5, then there is an equivalent instance (G', k) of* Tree Deletion Set *such that $|V(G')| \leq (k + 1)|V(G)|$ and $|E(G')| \leq |E(G)| + |V(G')|$.*

▶ **Theorem 5** (Bafna et al. [1])**.** *There is an $\mathcal{O}(\min\{|E(G)| \log |V(G)|, |V(G)|^2\})$ time algorithm that given a graph G that admits a feedback vertex set of size at most k outputs a feedback vertex set of G of size at most $2k$.*

3.2 A structural decomposition

In this subsection we decompose an instance (G, w, k) of wTDS to an equivalent instance (G', w', k') where $V(G')$ is partitioned into three sets B, T, and I, such that the size of B and T is polynomial in k and I is an independent set. In particular we obtain the following result.

▶ **Lemma 6.** *There is a polynomial time algorithm that given a semi-reduced instance (G, w, k) of* wTDS *either correctly decides that (G, w, k) is a no-instance or outputs an equivalent instance (G', w', k') and a partition of $V(G')$ into sets B, T, and I such that*

(i) *$|B| \leq 8k^2 + 2k$,*
(ii) *T induces a forest and $|T| \leq 240k^4 + 272k^3 + 65k^2 - 19k - 7$,*
(iii) *I is an independent set, and*
(iv) *for every $v \in I$, $N_{G'}(v) \subseteq B$, $|N_{G'}(v)| \leq 2k + 1$, and $N_{G'}(v)$ is a double clique.*

For an example of the structure of the graph G' obtained from Lemma 6, see Figure 1.

We split the proof of this lemma into several auxiliary lemmata. We start by identifying the set B.

▶ **Lemma 7.** *There is a polynomial time algorithm that given a semi-reduced instance (G, w, k) of* wTDS *either correctly decides that (G, w, k) is a no-instance or finds two sets F and $\widehat{Q}$ such that, denoting $B = F \cup \widehat{Q}$, the following holds.*

(i) *F is a feedback vertex set of G.*
(ii) *Each connected component of $G \setminus B$ has at most 2 neighbors in $\widehat{Q}$.*

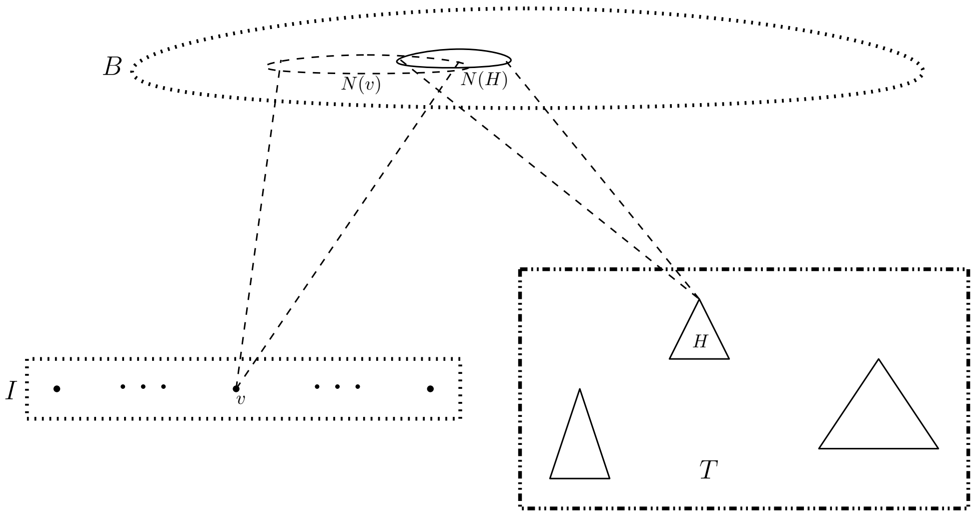


Figure 1 The vertex set of the graph G' is partitioned into a set B, a set T where every connected component H of T is a tree, and a set I. The set I induces an independent set and for every vertex $v \in I$, $N_{G'}(v) \subseteq B$ and $N_{G'}(v)$ induces a double clique.

(iii) *For every connected component $H \in \mathcal{C}(G \setminus B)$ and every vertex $y \in B$, $|N_G(y) \cap H| \leq 1$, that is, every vertex y of F and every vertex y of $\widehat{Q}$ have at most one neighbor in every connected component H of $G \setminus B$.*

(iv) $|B| \leq 8k^2 + 2k$.

Proof. First notice that every tree deletion set of G of weight at most k is also a feedback vertex set of G of size at most k in the underlying non-weighted graph. Thus, by applying Theorem 5 we may find in polynomial time a feedback vertex set F of G. If $|F| > 2k$, then output NO. Otherwise, $|F| \leq 2k$.

As the instance (G, w, k) is semi-reduced, Reduction Rule 4 is not applicable, and G does not contain an x-flower of order $k+1$ for any $x \in F$. Therefore, from Theorem 3, we get that for every $x \in F$ we can find in polynomial time a set $Q^x \subseteq V(G) \setminus \{x\}$ intersecting every cycle that goes through x in G and such that $|Q^x| \leq 2k$. Let $Q = \bigcup_{x \in F} Q^x$.

Let $\mathcal{C}(G \setminus F) = \{H_1, H_2, \ldots, H_l\}$ and note that, as F is a feedback vertex set of G, each $G[H_i]$ is a tree. From now on, without loss of generality we will assume that each $G[H_i]$, $i \in [l]$, is rooted at some vertex $v_i \in H_i$.

Let $Q_i = H_i \cap Q$, $i \in [l]$. In other words, Q_i denotes the set of vertices of H_i that are also vertices of Q, $i \in [l]$. Let also $\widehat{Q}_i = \textbf{LCA-closure}(Q_i)$, that is, let $\widehat{Q}_i$ denote the least common ancestor-closure of the set Q_i in the tree $G[H_i]$. Finally, let $\widehat{Q} = \bigcup_{i \in [l]} \widehat{Q}_i$ and note that $\widehat{Q} \cap F = \emptyset$.

Let us now prove that F and $\widehat{Q}$ have the claimed properties. First of all, F is a feedback vertex set by construction, proving (i). Second, since for each x in F we have $|Q^x| \leq 2k$, we have $|Q| \leq 4k^2$, and from Lemma 1 we get that $|\widehat{Q}| = |\bigcup_{i \in [l]} \widehat{Q}_i| = \sum_{i \in [l]} |\widehat{Q}_i| \leq 2 \sum_{i \in l} |Q_i| \leq 2|Q| \leq 8k^2$. Together with $|F| \leq 2k$ this proves (iv). Third, from the construction of $\widehat{Q}$ and from Lemma 1 we get the property (ii).

Let us now prove (iii). Let $y \in B$ and $H \in \mathcal{C}(G \setminus B)$ and assume to the contrary that $|N_G(y) \cap H| \geq 2$. Then, as $G[H]$ is connected, the graph $G[H \cup \{y\}]$ contains a cycle that goes through y. If $y \in F$, we get a contradiction to the facts that $G[H \cup \{y\}]$ is a subgraph of $G \setminus Q^y$ and the set Q^y intersects every cycle that goes through y. If $y \in \widehat{Q}$, we get a contradiction, since $G[H \cup \{y\}]$ is a subgraph of $G \setminus F$ (recall that $\widehat{Q} \cap F = \emptyset$) and $G \setminus F$ is acyclic. ◀

The next lemma shows that if B is as in the previous lemma, then the size of connected components in the rest of the graph is bounded.

▶ **Lemma 8** ($\star$). *If (G, w, k) and B are as in Lemma 7 and H is a connected component of $G \setminus B$, then $|H| \leq 12k + 7$.*

Let x, y be two vertices of B. We say that the pair $\{x, y\}$ is in $\mathcal{P}^{\leq k+1}$ if there are at most $k+1$ connected components H of $G \setminus B$ with $\{x, y\} \subseteq N_G(H)$ and that $\{x, y\}$ is in $\mathcal{P}^{\geq k+2}$ otherwise. Now we add to G a double edge between every pair in $\mathcal{P}^{\geq k+2}$ to obtain the graph $\widehat{G}$. The next lemma shows that the resulting instance is equivalent to the original one.

▶ **Lemma 9.** *The instance $(\widehat{G}, w, k)$, where $\widehat{G}$ is as defined above, is equivalent to (G, w, k).*

Proof. Let $\{x, y\} \in \mathcal{P}^{\geq k+2}$. Notice that each connected component H of $G \setminus B$ with $\{x, y\} \subseteq N_G(H)$ provides a separate path between x and y. Observe then that if neither x nor y belong to a tree deletion set D of G we need at least $k+1$ vertices to hit all the cycles, since otherwise there are at least two components $H_1, H_2 \in \mathcal{C}(G \setminus B)$ with $\{x, y\} \subseteq (N_G(H_1) \cap N_G(H_2))$ and $(H_1 \cup H_2) \cap D = \emptyset$ and thus the graph induced by $H_1 \cup H_2 \cup \{y, y'\}$ contains a cycle. This implies that (G, w, k) is a yes-instance if and only if at least one of the vertices x and y is contained in every tree deletion set of G of weight k. ◀

The following lemma shows that there are only few connected components of $G \setminus B$ having a neighborhood that is not a double clique in $\widehat{G}$.

▶ **Lemma 10.** *If (G, w, k) and B are as in Lemma 7 and $\widehat{G}$ as defined above, then there is a set $\mathcal{C}_T \subseteq \mathcal{C}(G \setminus B)$ such that*
(i) $|\mathcal{C}_T| \leq 20k^3 + 11k^2 - k - 1$,
(ii) *for every H in $\mathcal{C}(G \setminus B) \setminus \mathcal{C}_T$, we have $N_G(H)$ is a double clique in $\widehat{G}$ and $|N_G(H) \cap Q| \leq 1$.*

Proof. For $x, y \in B$ we denote $S(x, y) = \{H \in \mathcal{C}(G \setminus B) \mid \{x, y\} \subseteq N_G(H)\}$. Let us set $\mathcal{C}_T = \bigcup_{\{x,y\} \in \mathcal{P}^{\leq k+1}} S(x, y)$. Let us now assume that there is H in $\mathcal{C}(G \setminus B) \setminus \mathcal{C}_T$, and two vertices x and y in $N_G(H)$ that are not joined by a double edge. By construction of the graph $\widehat{G}$, this implies that $\{x, y\} \in \mathcal{P}^{\leq k+1}$. But this implies that H is in $\mathcal{C}_T$, a contradiction. Furthermore, for every $x, y \in \widehat{Q}$ we have $|S(x, y)| \leq 1$ as otherwise we would have a cycle in $G \setminus F$ and F is a feedback vertex set. Hence $\mathcal{C}_T$ satisfies (ii). It remains to prove (i).

Let us first mention that it is easy to see that $\mathcal{C}_T$ is of polynomial size. Indeed, we have $|\mathcal{C}_T| = |\bigcup_{\{x,y\} \in \mathcal{P}^{\leq k+1}} S(x, y)| \leq |B|^2(k+1) = \mathcal{O}(k^5)$. For the purpose of the more precise size bound let us distinguish three subsets of $\mathcal{C}_T$:

$$\begin{aligned}
\mathcal{T}^{FF} &= \bigcup\nolimits_{\{x,y\} \subseteq F \wedge \{x,y\} \in \mathcal{P}^{\leq k+1}} S(x, y) \\
\mathcal{T}^{QQ} &= \bigcup\nolimits_{\{x,y\} \subseteq \widehat{Q} \wedge \{x,y\} \in \mathcal{P}^{\leq k+1}} S(x, y) \\
\mathcal{T}^{FQ} &= \left(\bigcup\nolimits_{x \in F \wedge y \in \widehat{Q} \wedge \{x,y\} \in \mathcal{P}^{\leq k+1}} S(x, y) \right) \setminus \mathcal{T}^{QQ}
\end{aligned}$$

Obviously, $\mathcal{C}_T \subseteq (\mathcal{T}^{FF} \cup \mathcal{T}^{QQ} \cup \mathcal{T}^{FQ})$. Hence, to bound the size of $\mathcal{C}_T$ it is enough to bound the sizes of $\mathcal{T}^{FF}$, $\mathcal{T}^{QQ}$, and $\mathcal{T}^{FQ}$. Note that for every $\{x, y\} \in \mathcal{P}^{\leq k+1}$ we have $|S(x, y)| \leq k+1$. It follows that $|\mathcal{T}^{FF}| \leq \binom{|F|}{2}(k+1) \leq \binom{2k}{2}(k+1) = 2k^3 + k^2 - k$.

Next we claim that $|\mathcal{T}^{QQ}| \leq |\widehat{Q}| - 1 \leq 8k^2 - 1$. For every $x, y \in \widehat{Q}$ we have $|S(x, y)| \leq 1$ as otherwise we would have a cycle in $G \setminus F$ and F is a feedback vertex set. Let A_Q be the graph with vertex set $\widehat{Q}$ where two vertices in $\widehat{Q}$ are connected by an edge if and only if they are the neighbors of a component $H \in \mathcal{T}^{QQ}$ in $\widehat{Q}$. Hence, the number of edges of A_Q equals $|\mathcal{T}^{QQ}|$. We now work towards showing that A_Q is a forest. Indeed, assume to the contrary that there exists a cycle in A_Q. Then it is easy to see that we may find a cycle in the graph $\widehat{H}$ induced by the components in $\mathcal{T}^{QQ}$ which correspond to the edges of the cycle in A_Q and their neighborhood in $\widehat{Q}$. Recall that $\widehat{Q} \cap F = \emptyset$ and therefore $\widehat{H}$ is a subgraph of $G \setminus F$. This contradicts the fact that F is a feedback vertex set of G. Hence, A_Q is a forest and the claim follows.

For the upper bound on $\mathcal{T}^{FQ}$, for every $x \in F$ we partition the set $\widehat{Q}$ into two sets $R_x^{\leq 1}$ and $R_x^{\geq 2}$ in the following way.

$$\begin{aligned} R_x^{\leq 1} &= \{y \in \widehat{Q} \mid \text{there is at most 1 component } H \in \mathcal{T}^{FQ} \text{ such that } \{x, y\} \subseteq N_G(H)\} \\ R_x^{\geq 2} &= \{y \in \widehat{Q} \mid \{x, y\} \in \mathcal{P}^{\leq k+1} \text{ and there exist at least two distinct components} \\ &\qquad H_1, H_2 \in \mathcal{T}^{FQ} \text{ such that } \{x, y\} \subseteq N_G(H_1) \cap N_G(H_2)\}. \end{aligned}$$

Observe that $|\mathcal{T}^{FQ}| \leq \sum_{x \in F} \big(|R_x^{\leq 1}| + |R_x^{\geq 2}|(k+1)\big)$ and for every $x \in F$, it trivially holds that $|R_x^{\leq 1}| \leq |\widehat{Q}| \leq 8k^2$.

Moreover, we claim that for every $x \in F$, $|R_x^{\geq 2}| \leq k$. Indeed, assume to the contrary that $|R_x^{\geq 2}| \geq k + 1$ for some $x \in F$. Then there exist $k + 1$ vertices $y_i \in \widehat{Q}$, $i \in [k + 1]$, such that for every i there exist two connected components H_1^i and H_2^i in $\mathcal{T}^{FQ} \subseteq \mathcal{C}(G \setminus B) \setminus \mathcal{T}^{QQ}$ such that $\{x, y\} \subseteq N_G(H_1^i) \cap N_G(H_2^i)$. This implies that the graph induced by the vertex x, the vertices y_i, $i \in [k + 1]$, and the components H_1^i and H_2^i, $i \in [k + 1]$, contains an x-flower of order $k + 1$ (notice that, as none of the graphs belong to $\mathcal{T}^{QQ}$, they are pairwise disjoint). This is a contradiction to the fact that G is semi-reduced. Therefore, for every $x \in F$ we have $|R_x^{\geq 2}| \leq k$.

Alltogether, we have $|\mathcal{T}^{FQ}| \leq \sum_{x \in F} \big(8k^2 + k(k + 1)\big) \leq 18k^3 + 2k^2$ and $|\mathcal{C}_T| \leq |\mathcal{T}^{FF}| + |\mathcal{T}^{QQ}| + |\mathcal{T}^{FQ}| \leq (2k^3 + k^2 - k) + (8k^2 - 1) + (18k^3 + 2k^2) = 20k^3 + 11k^2 - k - 1$ proving (i). ◀

Let us denote $T = \bigcup_{H \in \mathcal{C}_T} H$. Note that by the properties of $\mathcal{C}_T$ we have $\mathcal{C}(\widehat{G} \setminus (B \cup T)) = \mathcal{C}(G \setminus B) \setminus \mathcal{C}_T$. Further, by Lemma 8 we have $|T| \leq |\mathcal{C}_T|(12k + 7)$ and, hence, by Lemma 10, $|T| \leq (20k^3 + 11k^2 - k - 1)(12k + 7) = 240k^4 + 272k^3 + 65k^2 - 19k - 7$.

We now prove that the components of $\mathcal{C}(G \setminus B)$ that are not in $\mathcal{C}_T$ behave as single vertices with respect to tree deletion sets.

▶ **Lemma 11** ($\star$). *If there exists a tree deletion set S of $\widehat{G}$ of weight at most k then there exists a tree deletion set $\widehat{S}$ of $\widehat{G}$ of weight at most k such that for every $H \in \mathcal{C}(\widehat{G} \setminus (B \cup T))$, either $H \subseteq \widehat{S}$ or $H \cap \widehat{S} = \emptyset$.*

Now, let G' be the graph obtained from $\widehat{G}$ after contracting every connected component H of $\widehat{G} \setminus (B \cup T)$ into a single vertex v_H and setting $w'(v_H) = \sum_{v \in H} w(v)$ and $w'(v) = w(v)$ for every $v \in (B \cup T)$. We also define I to be the set $V(G') \setminus (B \cup T)$. We now prove that such a contraction does not affect the instance.

▶ **Lemma 12** ($\star$). *If $\widehat{G}$, G', and w' are as defined above, then the instances $(\widehat{G}, w, k)$ and (G', w', k) are equivalent.*

Lemma 6 now follows directly from Lemmata 7–12.

▶ Remark. While it might be tempting to say that among a pair of vertices in $\mathcal{P}^{\geq k+2}$ a solution must remove exactly one, this is not the case. Though, clearly, some of the common neighbors of the pair remain untouched, they might be connected to the rest of the graph through other vertices of B. Hence it might be the case that both vertices of the pair are removed.

3.3 Results on Linear Equations

▶ **Lemma 13.** *Let $\mathbb{F}$ be a field. For every matrix $M \in \mathbb{F}^{m\times n}$ and positive integer k, there exists a submatrix $M' \in \mathbb{F}^{m'\times n}$ of M, where $m' \leq n(k+1)$, such that for every $x \in \mathbb{F}^n$ with $\mathbf{d}_H(M' \cdot x^T, \mathbf{0}^{m'}) \leq k$, $\mathbf{d}_H(M \cdot x^T, \mathbf{0}^m) = \mathbf{d}_H(M' \cdot x^T, \mathbf{0}^{m'})$. Furthermore, the matrix M' can be computed in time $\mathcal{O}(m \cdot n^{\omega-1}k)$, where ω is the matrix multiplication exponent ($\omega < 2.373$ [24]), assuming that the field operations take a constant time.*

Proof. In order to identify M' we identify $j_0 + 1 \leq k + 1$ (non-empty) submatrices $B_0, B_1, \ldots, B_{j_0}$ of M, each having at most n rows, in the following way: First, let B_0 be a minimal submatrix of M whose rows span all the rows of M, that is, let B_0 be a base of the vector space generated by the rows of M, and let also M_0 be the submatrix obtained from M after removing the rows of B_0. We identify the rest of the matrices inductively as follows: For every $i \in [k]$, if M_{i-1} is not the empty matrix we let B_i be a minimal submatrix of M_{i-1} whose rows span all the rows of M_{i-1} and finally we let M_i be the matrix occurring from M_{i-1} after removing the rows of B_i.

We now define the submatrix M' of M. Let $j_0 \leq k$ be the greatest integer for which M_{j_0-1} is not the empty matrix. Let M' be the matrix consisting of the union of the rows of the (non-empty) matrices B_0 and B_i, $i \in [j_0]$. As the rank of the matrices M, M_i, $i \in [j_0]$, is upper bounded by n, the matrices B_0, B_i, $i \in [j_0]$, have at most n rows each, and therefore M' has at most $n(j_0 + 1) \leq n(k + 1)$ rows. Observe that if $j_0 < k$ then the union of the rows of the non-empty matrices B_0, B_i, $i \in [j_0]$, contains all the rows of M and thus we may assume that $M' = M$ and the lemma trivially holds. Hence, it remains to prove the lemma for the case where $j_0 = k$, and therefore M' consists of the union of the matrices B_0, B_i, $i \in [k]$. As it always holds that $\mathbf{d}_H(M \cdot x^T, \mathbf{0}) \geq \mathbf{d}_H(M' \cdot x^T, \mathbf{0})$ it is enough to prove that for every $x \in \mathbb{F}^n$ for which $\mathbf{d}_H(M' \cdot x^T, \mathbf{0}) \leq k$, $\mathbf{d}_H(M \cdot x^T, \mathbf{0}) \leq \mathbf{d}_H(M' \cdot x^T, \mathbf{0})$. Thus, it is enough to prove that for every row r of the matrix M'' obtained from M after removing the rows of M', it holds that $\mathbf{d}_H(r \cdot x^T, \mathbf{0}) = 0$. Towards this goal let $x \in \mathbb{F}^n$ be a vector such that $\mathbf{d}_H(M' \cdot x^T, \mathbf{0}) \leq k$. From the Pigeonhole Principle there exists an i_0 such that $\mathbf{d}_H(B_{i_0} \cdot x^T, \mathbf{0}) = 0$, that is, if $r_1, r_2, \ldots, r_{|B_{i_0}|}$ are the rows of B_{i_0} then $r_j \cdot x^T = 0$, for every $j \in [|B_{i_0}|]$. Recall however that the row r of M'' is spanned by the rows $r_1, r_2, \ldots, r_{|B_{i_0}|}$ of B_{i_0}. Therefore, there exist $\lambda_j \in \mathbb{F}$, $j \in [|B_{i_0}|]$, such that $r = \sum_{j\in[|B_{i_0}|]} \lambda_j r_j$. It follows that $r \cdot x^T = \sum_{j\in[|B_{i_0}|]} \lambda_j (r_j \cdot x^T) = 0$ and therefore $\mathbf{d}_H(r \cdot x^T, \mathbf{0}) = 0$. This implies that $\mathbf{d}_H(M \cdot x^T, \mathbf{0}) \leq \mathbf{d}_H(M' \cdot x^T, \mathbf{0})$. Finally, for a rectangular matrix of size $d \times r$, $d \leq r$, Ibarra et al. [12] give an algorithm that computes a maximal independent set of rows (a row basis) in $\mathcal{O}(d^{\omega-1}r)$ time. By running this algorithm $k + 1$ times we can find the matrix M' in $\mathcal{O}(mn^{\omega-1}k)$ time and this completes the proof of the lemma. ◀

▶ **Lemma 14.** *Let $\mathbb{F}$ be a field. There exists an algorithm that given a set $\mathcal{S}$ of linear equations over $\mathbb{F}$ on n variables and an integer k outputs a set $\mathcal{S}' \subseteq \mathcal{S}$ of at most $(n+1)(k+1)$ linear equations over $\mathbb{F}$ such that any assignment of the variables that violates at most k linear equations of $\mathcal{S}'$ satisfies all the linear equations of $\mathcal{S} \setminus \mathcal{S}'$. Moreover, the running time of the algorithm is $\mathcal{O}(|\mathcal{S}|n^{\omega-1}k)$, assuming that the field operations take a constant time.*

Proof. Let $x_1, x_2, \dots, x_n$ denote the n variables and α_{ij} denote the coefficient of x_j in the i-th linear equation of S, $i \in [|\mathcal{S}|]$, $j \in [n]$. Let also $\alpha_{i(n+1)}$ denote the constant term of the i-th linear equation of $\mathcal{S}$. In other words, the i-th equation of $\mathcal{S}$ is denoted as $\alpha_{i1}x_1 + \alpha_{i2}x_2 + \cdots + \alpha_{in}x_n + \alpha_{i(n+1)} = 0$. Finally, let M be the matrix where the j-element of the i-th row is α_{ij}, $i \in [|\mathcal{S}|]$, $j \in [n+1]$. From Lemma 13, it follows that for every positive integer k there exists a submatrix M' of M with at most $(n+1)(k+1)$ rows and $n+1$ columns such that for every $x \in \mathbb{F}^{n+1}$ for which $\mathbf{d}_H(M' \cdot x^T, \mathbf{0}) \leq k$, $\mathbf{d}_H(M \cdot x^T, \mathbf{0}) = \mathbf{d}_H(M' \cdot x^T, \mathbf{0})$ and M' can be computed in time $\mathcal{O}(|\mathcal{S}|n^{\omega-1}k)$. Let $\mathcal{S}'$ be the set of linear equations that correspond to the rows of M'. Let then $x_i = \beta_i$, $\beta_i \in \mathbb{F}$, $i \in [n]$, be an assignment that does not satisfy at most k of the equations of $\mathcal{S}'$. This implies that $\mathbf{d}_H(M' \cdot z, \mathbf{0}) \leq k$, where $z = (\beta_1, \beta_2, \dots, \beta_n, 1)^T$. Again, from Lemma 13, we get that $\mathbf{d}_H(M \cdot z, \mathbf{0}) = \mathbf{d}_H(M' \cdot z, \mathbf{0})$. Thus, the above assignment satisfies all the linear equations of $\mathcal{S} \setminus \mathcal{S}'$. ◀

3.4 The Main Theorem

In this subsection by combining the structural decomposition of Subsection 3.2 and Lemma 14 from Subsection 3.3 we obtain a kernel for wTDS of size $\mathcal{O}(k^4)$.

▶ **Theorem 15.** wTDS *admits a kernel of size* $\mathcal{O}(k^4)$ *and* $\mathcal{O}(k^4 \log k)$ *bits.*

Proof. Let (G, w, k) be an instance of wTDS. Without loss of generality we may assume that it is semi-reduced, G is connected, and that, from Lemma 6, $V(G)$ can be partitioned into three sets B, T, and I satisfying the conditions of Lemma 6. Note that, as G is connected, every vertex of I has at least one neighbor in B. We construct an instance (G', w', k) of wTDS in the following way. Let p be a prime number such that $|B| < p < 2|B|$. Such a prime number exists by a Bertrand's postulate (proved by Chebyshev in 1850). Let $\mathbb{F} = \mathbb{GF}(p)$, that is, the Galois field of order p. It takes at most $O(|B|^2) = O(k^4)$ time to find p and the multiplicative inverses in $\mathbb{F}$.

Let $I = \{v_i \mid i \in [|I|]\}$ and $B = \{u_j \mid j \in [|B|]\}$. We assign an $\mathbb{F}$-variable x_j to u_j, $j \in [|B|]$, and a linear equation l_i over $\mathbb{F}$ to v_i, $i \in [|I|]$, where l_i is the equation $\sum_{j \in [|B|]} \alpha_{ij}x_j - 1 = 0$ and $\alpha_{ij} = 1$ if $u_j \in N_G(v_i)$ and 0 otherwise. Let $\mathcal{L} = \{l_i \mid i \in [|I|]\}$ and $\mathcal{L}'$ be the subset of $\mathcal{L}$ obtained from Lemma 14. Let also $I' = \{v_p \in I \mid l_p \in \mathcal{L}'\}$ and $G' = G[B \cup T \cup I']$. Finally, let $w' = w|_{B \cup T \cup I'}$. We now prove that (G', w', k) is equivalent to (G, w, k).

We first prove that if (G, w, k) is a yes-instance then so is (G', w', k). Let S be a tree deletion set of G of weight at most k. Then $G \setminus S$ is a tree and, as for every vertex $v \in I \setminus S$, $N_G(v)$ is a double clique, v has degree exactly 1 in $G \setminus S$. Therefore, the graph obtained from $G \setminus S$ after removing $(I \setminus I')$ is still a tree. This implies that $S \setminus (I \setminus I')$ is a tree deletion set of G' of weight at most k and (G', w', k) is a yes-instance.

Let now (G', w', k) be a yes-instance and S be a tree deletion set of G' of weight at most k. We claim that there exist at most k vertices in I' whose neighborhood lies entirely in S. Indeed, assume to the contrary that there exist at least $k+1$ vertices of I' whose neighborhood lies entirely in S. Let J be the set of those vertices. Notice that for every vertex $v \in I'$, if $N_{G'}(v) \subseteq S$, then either $v \in S$ or $I' \setminus \{v\} \subseteq S$. Notice that if $J \subseteq S$, then S has weight at least $k+1$, a contradiction. Therefore, there exists a vertex $u \in J$ that is not contained in S. Then $I' \setminus \{u\} \subseteq S$. Moreover, recall that u has at least one neighbor z in B and from the hypothesis z is contained in S. Therefore $(I' \setminus \{u\}) \cup \{z\} \subseteq S$. As $|I'| \geq |J| = k+1$, it follows that $|I' \setminus \{u\}| \geq k$. Furthermore, recall that $B \cap I' = \emptyset$. Thus, $|S| \geq k+1$, a contradiction to the fact that S has weight at most k. Therefore, there exist at most k vertices of I' whose neighborhood is contained entirely in S. For every $j \in [|B|]$, let $x_j = \beta_j$, where $\beta_j = 0$ if $u_j \in S$ and 1 otherwise. Then there exist at most k linear equations in $\mathcal{L}'$ which are not satisfied by the above assignment. However, from the choice of $\mathcal{L}'$ all the linear equations in $\mathcal{L} \setminus \mathcal{L}'$ are satisfied and therefore, for every vertex u in $I \setminus I'$ we have $|N_G(u) \setminus S| \equiv 1 \pmod{p}$. Since $p > |B|$ this implies that u has exactly one neighbor in $G \setminus S$. Thus $G \setminus S$ is a tree and hence, S is a tree deletion set of G as well.

Notice that $V(G') = B \cup T \cup I'$, where $|I'| \leq 8k^3 + 10k^2 + 3k + 1$ (Lemma 14) and therefore $|V(G')| = \mathcal{O}(k^4)$. It is also easy to see that $|E(G')| = \mathcal{O}(k^4)$. Indeed, notice first that as the set I' is an independent set there are no edges between its vertices. Moreover, from Lemma 6 there are no edges between the vertices of the set I' and the set T. Observe that, from the construction of I and subsequently of I', Lemma 6 implies that every vertex of I' has at most $2k + 2$ neighbors in B. As $|I'| \leq 8k^3 + 10k^2 + 3k + 1$ there exist $\mathcal{O}(k^4)$ edges between the vertices of I' and the vertices of B. Notice that from (2) of Lemma 6, T induces a forest and thus there exist at most $\mathcal{O}(k^4)$ edges between its vertices. Moreover, from (1) of Lemma 6, again there exist $\mathcal{O}(k^4)$ edges between the vertices of B. It remains to show that there exist $\mathcal{O}(k^4)$ edges with one endpoint in B and one endpoint in T. Recall first that every connected component has at most 2 neighbors in $\widehat{Q}$. Therefore, there exist at most $2k + 2$ edges between every connected component of $\mathcal{C}_T$ and B. Moreover, from Lemma 10 we obtain that $\mathcal{C}_T$ contains $\mathcal{O}(k^3)$ connected components. Therefore, there exist $\mathcal{O}(k^4)$ edges with one endpoint in B and one endpoint in T. Thus, wTDS has a kernel of $\mathcal{O}(k^4)$ vertices and edges. Finally, from Reduction Rule 5, the weight of every vertex is upper bounded by $k + 1$ and thus, it can be encoded using $\log(k + 1)$ bits resulting to a kernel of wTDS with $\mathcal{O}(k^4 \log k)$ bits. ◀

From Lemma 4 we immediately get the following corollary.

▶ **Corollary 16.** Tree Deletion Set *has a kernel with* $\mathcal{O}(k^5)$ *vertices and edges.*

References

1 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999.

2 Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010.

3 Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In *Parameterized and Exact Computation – IWPEC*, volume 4169 of *LNCS*, pages 192–202, 2006.

4 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In *Algorithm Theory – SWAT 2010*, volume 6139 of *LNCS*, pages 93–104, 2010.

5 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

6 Robert Crowston, Michael R. Fellows, Gregory Gutin, Mark Jones, Frances A. Rosamond, Stéphan Thomassé, and Anders Yeo. Simultaneously satisfying linear equations over $\mathbb{F}_2$: MaxLin2 and Max-r-Lin2 parameterized above average. In *Foundations of Software Technology and Theoretical Computer Science – FSTTCS 2011*, volume 13 of *LIPIcs*, pages 229–240, 2011.

7 Robert Crowston, Gregory Gutin, Mark Jones, Eun Jung Kim, and Imre Z. Ruzsa. Systems of linear equations over $\mathbb{F}_2$ and problems parameterized above average. In *Algorithm Theory – SWAT 2010*, volume 6139 of *LNCS*, pages 164–175, 2010.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

9 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *Foundations of Computer Science – FOCS 2012*, pages 470–479, 2012.

10 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

11 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

12 Oscar H. Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. Algorithms*, 3(1):45–56, 1982.

13 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

14 Stefan Kratsch. Polynomial kernelizations for MIN $F^{+}\Pi$ and MAX NP. *Algorithmica*, 63(1-2):532–550, 2012.

15 Stefan Kratsch and Magnus Wahlström. Preprocessing of Min Ones problems: A dichotomy. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6–10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 653–665. Springer, 2010.

16 Stefan Kratsch and Magnus Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In *Symposium on Discrete Algorithms – SODA 2012*, pages 94–103, 2012.

17 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Foundations of Computer Science – FOCS 2012*, pages 450–459, 2012.

18 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization – preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *LNCS*, pages 129–161, 2012.

19 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

20 Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications).* Oxford University Press, USA, March 2006.

21 Venkatesh Raman, Saket Saurabh, and Ondřej Suchý. An FPT algorithm for tree deletion set. In *Algorithms and Computation – WALCOM 2013*, volume 7748 of *LNCS*, pages 286–297, 2013.

22 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010.

23 Magnus Wahlström. Abusing the Tutte matrix: An algebraic instance compression for the K-set-cycle problem. In *Symposium on Theoretical Aspects of Computer Science – STACS 2013*, volume 20 of *LIPIcs*, pages 341–352, 2013.

24 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012*, pages 887–898. ACM, 2012.

25 Mihalis Yannakakis. The effect of a connectivity requirement on the complexity of maximum subgraph problems. *J. ACM*, 26(4):618–630, 1979.

Editing to Eulerian Graphs*

Konrad K. Dabrowski[1], Petr A. Golovach[2], Pim van 't Hof[2], and Daniël Paulusma[1]

1 **School of Engineering and Computing Sciences, Durham University, Science Laboratories, South Road, Durham DH1 3LE, United Kingdom**
`{konrad.dabrowski,daniel.paulusma}@durham.ac.uk`

2 **Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway**
`{petr.golovach,pim.vanthof}@ii.uib.no`

Abstract

We investigate the problem of modifying a graph into a connected graph in which the degree of each vertex satisfies a prescribed parity constraint. Let ea, ed and vd denote the operations edge addition, edge deletion and vertex deletion respectively. For any $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$, we define Connected Degree Parity Editing(S) (CDPE(S)) to be the problem that takes as input a graph G, an integer k and a function $\delta\colon V(G) \to \{0, 1\}$, and asks whether G can be modified into a connected graph H with $d_H(v) \equiv \delta(v) \pmod 2$ for each $v \in V(H)$, using at most k operations from S. We prove that

- if $S = \{\mathsf{ea}\}$ or $S = \{\mathsf{ea}, \mathsf{ed}\}$, then CDPE($S$) can be solved in polynomial time;
- if $\{\mathsf{vd}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$, then CDPE($S$) is NP-complete and W[1]-hard when parameterized by k, even if $\delta \equiv 0$.

Together with known results by Cai and Yang and by Cygan, Marx, Pilipczuk, Pilipczuk and Schlotter, our results completely classify the classical and parameterized complexity of the CDPE(S) problem for all $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$. We obtain the same classification for a natural variant of the CDPE(S) problem on directed graphs, where the target is a weakly connected digraph in which the difference between the in- and out-degree of every vertex equals a prescribed value.

As an important implication of our results, we obtain polynomial-time algorithms for Eulerian Editing problem and its directed variant. To the best of our knowledge, the only other natural non-trivial graph class $\mathcal{H}$ for which the $\mathcal{H}$-Editing problem is known to be polynomial-time solvable is the class of split graphs.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Eulerian graphs, graph editing, polynomial algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.97

1 Introduction

Graph modification problems play a central role in algorithmic graph theory, partly due to the fact that they naturally arise in numerous practical applications. A graph modification problem takes as input a graph G and an integer k, and asks whether G can be modified

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 267959 and from EPSRC Grant EP/K025090/1.

into a graph belonging to a prescribed graph class $\mathcal{H}$, using at most k operations of a certain type. The most common operations that are considered in this context are edge additions ($\mathcal{H}$-Completion), edge deletions ($\mathcal{H}$-Edge Deletion), vertex deletions ($\mathcal{H}$-Vertex Deletion), and a combination of edge additions and edge deletions ($\mathcal{H}$-Editing). The intensive study of graph modification problems has produced a plethora of classical and parameterized complexity results (see e.g. [1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 14, 15, 16, 17, 19, 20]).

An undirected graph is Eulerian if it is connected and every vertex has even degree, while a directed graph is Eulerian if it is strongly connected[1] and balanced, i.e. the in-degree of every vertex equals its out-degree. Eulerian graphs form a well-known graph class both within algorithmic and structural graph theory. Several groups of authors have investigated the problem of deciding whether a given graph can be made Eulerian using a small number of operations. Boesch et al. [1] presented a polynomial-time algorithm for Eulerian Completion, and Cai and Yang [4] showed that the problems Eulerian Vertex Deletion and Eulerian Edge Deletion are NP-complete [4]. When parameterized by k, it is known that Eulerian Vertex Deletion is W[1]-hard [4], while Eulerian Edge Deletion is fixed-parameter tractable [7]. Cygan et al. [7] showed that the classical and parameterized complexity results for Eulerian Vertex Deletion and Eulerian Edge Deletion also hold for the directed variants of these problems.

Our Contribution. We generalize, extend and complement known results on graph modification problems dealing with Eulerian graphs and digraphs. The main contribution of this paper consists of two non-trivial polynomial-time algorithms: one for solving the Eulerian Editing problem, and one for solving the directed variant of this problem. Given the aforementioned NP-completeness result for Eulerian Edge Deletion and the fact that $\mathcal{H}$-Editing is NP-complete for almost all natural graph classes $\mathcal{H}$ [2, 20], we find it particularly interesting that Eulerian Editing turns out to be polynomial-time solvable. To the best of our knowledge, the only other natural non-trivial graph class $\mathcal{H}$ for which $\mathcal{H}$-Editing is known to be polynomial-time solvable is the class of split graphs [13].

In fact, our polynomial-time algorithms are implications of two more general results. In order to formally state these results, we need to introduce some terminology. Let ea, ed and vd denote the operations edge addition, edge deletion and vertex deletion, respectively. For any set $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$ and non-negative integer k, we say that a graph G can be (S,k)-*modified* into a graph H if H can be obtained from G by using at most k operations from S. We define the following problem for every $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$:

CDPE(S):	Connected Degree Parity Editing(S)
Instance:	A graph G, an integer k and a function $\delta\colon V(G) \to \{0,1\}$.
Question:	Can G be (S,k)-modified into a connected graph H with $d_H(v) \equiv \delta(v) \pmod 2$ for each $v \in V(H)$?

Inspired by the work of Cygan et al. [7] on directed Eulerian graphs, we also study a natural directed variant of the CDBE(S) problem. Denoting the in- and out-degree of a vertex v in a digraph G by $d^-_G(v)$ and $d^+_G(v)$, respectively, we define the following problem for every $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$:

[1] Replacing "strongly connected" by "weakly connected" yields an equivalent definition of Eulerian digraphs, as it is well-known that a balanced digraph is strongly connected if and only it is weakly connected (see e.g. [7]).

Table 1 A summary of the results for CDPE(S) and CDBE(S). All results are new except those for which a reference is given. The number of allowed operations k is the parameter in the parameterized results, and if a parameterized result is stated, then the corresponding problem is NP-complete.

S	CDPE(S)	CDBE(S)
ea, ed	P	P
ea	P	P
ed	FPT [7]	FPT [7]
vd	W[1]-hard [4]	W[1]-hard [7]
ea, vd	W[1]-hard	W[1]-hard
ed, vd	W[1]-hard	W[1]-hard
ea, ed, vd	W[1]-hard	W[1]-hard

CDBE(S):	CONNECTED DEGREE BALANCE EDITING(S)
Instance:	A digraph G, an integer k and a function $\delta\colon V(G) \to \mathbb{Z}$.
Question:	Can G be (S,k)-modified into a weakly connected digraph H with $d_H^+(v) - d_H^-(v) = \delta(v)$ for each $v \in V(H)$?

In Section 3, we prove that CDPE(S) can be solved in polynomial time when $S = \{\mathsf{ea}\}$ and when $S = \{\mathsf{ea}, \mathsf{ed}\}$. The first of these two results extends the aforementioned polynomial-time result by Boesch et al. [1] on EULERIAN COMPLETION and the second yields the first polynomial-time algorithm for EULERIAN EDITING, as these problems are equivalent to CDPE($\{\mathsf{ea}\}$) and CDPE($\{\mathsf{ea}, \mathsf{ed}\}$), respectively, when we set $\delta \equiv 0$. The complexity of the problem drastically changes when vertex deletion is allowed: we prove that for every subset $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$ with $\mathsf{vd} \in S$, the CDPE(S) problem is NP-complete and W[1]-hard with parameter k, even when $\delta \equiv 0$. This complements results by Cai and Yang [4] stating that CDPE(S) is NP-complete and W[1]-hard with parameter k when $S = \{\mathsf{vd}\}$ and $\delta \equiv 0$ or $\delta \equiv 1$. Our results, together with the aforementioned results due to Cygan et al. [7][2] and Cai and Yang [4], yield a complete classification of both the classical and the parameterized complexity of CDPE(S) for all $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$; see the middle column of Table 1.

In Section 4, we use different and more involved arguments to classify the classical and parameterized complexity of the CDBE(S) problem for all $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$. Interestingly, the classification we obtain for CDBE(S) turns out to be identical to the one we obtained for CDPE(S). In particular, our proof of the fact that CDBE(S) is polynomial-time solvable when $S = \{\mathsf{ea}\}$ and $S = \{\mathsf{ea}, \mathsf{ed}\}$ implies that the directed variants of EULERIAN COMPLETION and EULERIAN EDITING are not significantly harder than their undirected counterparts. All results on CDBE(S) are summarized in the right column of Table 1.

We would like to emphasize that there are no obvious hardness reductions between the different problem variants. The parameter k in the problem definitions represents the budget for all operations in total; adding a new operation to S may completely change the problem, as there is no way of forbidding its use. Hence, our polynomial-time algorithms for CDPE($\{\mathsf{ea}, \mathsf{ed}\}$) and CDBE($\{\mathsf{ea}, \mathsf{ed}\}$) do not generalize the polynomial-time algorithms for CDPE($\{\mathsf{ea}\}$) and CDBE($\{\mathsf{ea}\}$), and as such require significantly different arguments. In

[2] The FPT-results by Cygan et al. [7] only cover CDPE($\{\mathsf{ed}\}$) and CDBE($\{\mathsf{ed}\}$) when $\delta \equiv 0$, but it can easily be seen that their results carry over to CDPE($\{\mathsf{ed}\}$) and CDBE($\{\mathsf{ed}\}$) for any function δ.

particular, our main result, stating that Eulerian Editing is polynomial-time solvable, is not a generalization of the fact that Eulerian Completion is polynomial-time solvable and stands in no relation to the FPT-result by Cygan et al. [7] for Eulerian Edge Deletion.

We end this section by mentioning two similar graph modification frameworks in the literature that formed a direct motivation for the framework defined in this paper. Mathieson and Szeider [17] considered the Degree Constraint Editing(S) problem, which is that of testing whether a graph G can be k-modified into a graph H in which the degree of every vertex belongs to some list associated with that vertex. They classified the parameterized complexity of this problem for all $S \subseteq \{\mathsf{ea}, \mathsf{ed}, \mathsf{vd}\}$. Golovach [11] performed a similar study where the resulting graph must in addition be connected.

2 Preliminaries

We consider finite graphs $G = (V, E)$ that may be undirected or directed; in the latter case we will always call them digraphs. All our undirected graphs will be without loops or multiple edges; in particular, this is the case for both the input and the output graph in every undirected problem we consider. Similarly, for every directed problem that we consider, we do not allow the input or output digraph to contain multiple arcs. In our proofs we will also make use of *directed multigraphs*, which are digraphs that are permitted to have multiple arcs.

We denote an edge between two vertices u and v in a graph by uv. We denote an arc between two vertices u and v by (u, v), where u is the *tail* of (u, v) and v is the *head*. The disjoint union of two graphs G_1 and G_2 is denoted $G_1 + G_2$. The complete graph on n vertices is denoted K_n and the complete bipartite graph with classes of size s and t is denoted $K_{s,t}$.

Let $G = (V, E)$ be a graph or a digraph. Throughout the paper we assume that $n = |V|$ and $m = |E|$. For $U \subseteq V$, we let $G[U]$ be the graph (digraph) with vertex set U and an edge (arc) between two vertices u and v if and only if this is the case in G; we say that $G[U]$ is *induced by* U. We write $G - U = G[V \setminus U]$. For $E' \subseteq E$, we let $G(E')$ be the graph (digraph) with edge (arc) set E' whose vertex set consists of the end-vertices of the edges in E'; we say that $G(E')$ is *edge-induced by* E'. Let S be a set of (ordered) pairs of vertices of G. We let $G - S$ be the graph (digraph) obtained by deleting all edges (arcs) of $S \cap E$ from G, and we let $G + S$ be the graph (digraph) obtained by adding all edges (arcs) of $S \setminus E$ to G. We may write $G - e$ or $G + e$ if $S = \{e\}$.

Let $G = (V, E)$ be a graph. A component of G is a maximal connected subgraph of G. The complement of G is the graph $\overline{G} = (V, \overline{E})$ with vertex set V and an edge between two distinct vertices u and v if and only if $uv \notin E$. A *matching* M in G is a set of edges, in which no edge has a common end-vertex with some other edge. For a vertex $v \in V$, we let $N_G(v) = \{u \mid uv \in E\}$ denote its *(open) neighbourhood*. The *degree* of v is denoted $d_G(v) = |N_G(v)|$. The graph G is *even* if all its vertices have even degree, and it is *Eulerian* if it is even and connected. We say that a set $D \subseteq E$ is an *edge cut* in G if G is connected but $G - D$ is not. An edge cut of size 1 is called a *bridge* in G.

Let $G = (V, E)$ be a digraph. If (u, v) is an arc, then (v, u) is the *reverse* of this arc. For a subset $F \subseteq E$, we let $F^R = \{(u, v) | (v, u) \in F\}$ denote the set of arcs whose reverse is in F. The *underlying* graph of G is the undirected graph with vertex set V where two vertices $u, v \in V$ are adjacent if and only if (u, v) or (v, u) is an arc in G. We say that G is *(weakly) connected* if its underlying graph is connected. A *component* of G is a connected component of its underlying graph. An arc $a \in E$ is a *bridge* in G if it is a bridge in the underlying graph of G. A vertex u is an *in-neighbour* or *out-neighbour* of a vertex v if $(u, v) \in E$ or

$(v,u) \in E$, respectively. Let $N_G^-(v) = \{u \mid (u,v) \in E\}$ and $N_G^+(v) = \{u \mid (v,u) \in E\}$, where we call $d_G^-(v) = |N_G^-(v)|$ and $d_G^+(v) = |N_G^+(v)|$ the *in-degree* and *out-degree* of v, respectively. A vertex $v \in V$ is *balanced* if $d_G^+(v) = d_G^-(v)$. Recall that G is *Eulerian* if it is connected and *balanced*, that is, the out-degree of every vertex is equal to its in-degree.

Let $G = (V,E)$ be a graph and let $T \subseteq V$. A subset $J \subseteq E$ is a *T-join* if the set of odd-degree vertices in $G(J)$ is precisely T. If G is connected and $|T|$ is even then G has at least one T-join. In Section 3 we need to find a *minimum* T-join, that is, one of minimum size. We use the following result of Edmonds and Johnson [9] to do so.

▶ **Lemma 1** ([9]). *Let $G = (V,E)$ be a graph, and let $T \subseteq V$. Then a minimum T-join (if one exists) can be found in $O(n^3)$ time.*

Lemma 1 was used by Cygan et al. [7] to solve $\mathcal{H}$-Edge Deletion in polynomial time when $\mathcal{H}$ is the class of even graphs. It would immediately yield a polynomial-time algorithm for CDPE($\{\mathsf{ed}\}$) if we dropped the connectivity condition.

We need a variant of Lemma 1 for digraphs in Section 4. Let $G = (V,E)$ be a directed multigraph and let $f : T \to \mathbb{Z}$ be a function for some $T \subseteq V$. A multiset $E' \subseteq E$ with $T \subseteq V(G(E'))$ is a *directed f-join* in G if the following two conditions hold: $d^+_{G(E')}(v) - d^-_{G(E')}(v) = f(v)$ for every $v \in T$ and $d^+_{G(E')}(v) - d^-_{G(E')}(v) = 0$ for every $v \in V(G(E')) \setminus T$. A directed f-join is *minimum* if it has minimum size. The next lemma was used by Cygan et al. [7] to solve $\mathcal{H}$-Edge Deletion in polynomial time when $\mathcal{H}$ is the class of balanced digraphs; it would also yield a polynomial-time algorithm for CDBE($\{\mathsf{ed}\}$) if we dropped the connectivity condition.

▶ **Lemma 2** ([7]). *Let $G = (V,E)$ be a directed multigraph and $f : T \to \mathbb{Z}$ be a function for some $T \subseteq V$. A minimum directed f-join F (if one exists) can be found in $O(nm \log n \log\log m)$ time. Moreover, F consists of mutually arc-disjoint directed paths from vertices u with $f(u) > 0$ to vertices v with $f(v) < 0$.*

3 Connected Degree Parity Editing

We will show that CDPE(S) is polynomial-time solvable if $S = \{\mathsf{ea}\}$ or $S = \{\mathsf{ea}, \mathsf{ed}\}$ and that it is NP-complete and W[1]-hard with parameter k if $\mathsf{vd} \in S$.

First, let $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$. Let (G, δ, k) be an instance of CDPE(S) with $G = (V,E)$. Let A be a set of edges not in G, and let D be a set of edges in G, with $D = \emptyset$ if $S = \{\mathsf{ea}\}$. We say that (A, D) is a *solution* for (G, δ, k) if its *size* $|A| + |D| \leq k$, the congruence $d_H(u) \equiv \delta(u) \pmod 2$ holds for every vertex u and the graph $H = G + A - D$ is connected; if H is not connected then (A, D) is a *semi-solution* for (G, δ, k). If $S = \{\mathsf{ea}\}$ we may denote the solution by A rather than (A, D) (since $D = \emptyset$). We consider the optimization version for CDPE(S). The input is a pair (G, δ), and we aim to find the minimum k such that (G, δ, k) has a solution (if one exists). We call such a solution *optimal* and denote its size by $opt_S(G, \delta)$. We say that a (semi)-solution for (G, δ, k) is also a (semi)-solution for (G, δ). If (G, δ, k) has no solution for any value of k, then (G, δ) is a *no-instance* of CDPE(S) and $opt_S(G, \delta) = +\infty$.

Let $T = \{v \in V \mid d_G(v) \not\equiv \delta(v) \pmod 2\}$. Define $G_S = K_n$ if $S = \{\mathsf{ea}, \mathsf{ed}\}$ and $G_S = \overline{G}$ if $S = \{\mathsf{ea}\}$. Note that if $S = \{\mathsf{ea}\}$ then G_S contains no edges of G, so in this case any T-join in G_S can only contain edges in $E(\overline{G})$. The following key lemma is an easy observation.

▶ **Lemma 3.** *Let $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$. Let (G, δ) be an instance of* CDPE(S) *and $A \subseteq E(\overline{G})$, $D \subseteq E(G)$. Then (A, D) is a semi-solution of* CDPE(S) *if and only if $A \cup D$ is a T-join in G_S.*

We extend the result of Boesch et al. [1] for $\delta \equiv 0$ to arbitrary δ. Our proof is based around similar ideas but we also had to do some further analysis. The main difference in the two proofs is the following. If $\delta \equiv 0$ then none of the added edges in a solution will be a bridge in the modified graph (as the number of vertices of odd degree in a graph is always even). However this is no longer true for arbitrary δ and extra arguments are needed. We omit the proof of our result.

▶ **Theorem 4.** *Let $S = \{\mathsf{ea}\}$. Then* CDPE(S) *can be solved in $O(n^3)$ time.*

We are now ready to present the main result of this section. Recall that proving this result requires significantly different arguments than the ones used in the proof of Theorem 4. Let $S = \{\mathsf{ea}, \mathsf{ed}\}$ and let (G, δ) be an instance of CDPE(S). If F is a T-join in $G_S = K_n$, let $D = F \cap E(G)$ and $A = F \setminus D$. Then by Lemma 3, (A, D) is a semi-solution. Note that if F is a minimum T-join in G_S then it is a matching in which every vertex of T is incident to precisely one edge of F, so $|F| = \frac{1}{2}|T|$. We will show how this allows us to calculate $opt_S(G, \delta)$ directly from the structure of G, without having to find a T-join. We will also show that there are only trivial no-instances for this problem.

▶ **Theorem 5.** *Let $S = \{\mathsf{ea}, \mathsf{ed}\}$. Then* CDPE($S$) *can be solved in $O(n + m)$ time and an optimal solution (if one exists) can be found in $O(n^3)$ time.*

Proof. Let $S = \{\mathsf{ea}, \mathsf{ed}\}$ and let (G, δ) be an instance of CDPE(S). By Lemma 3, we may assume that $|T|$ is even, otherwise (G, δ) is a no-instance. If $G = K_2$ and $T = V(G)$, or $G = K_1 + K_1$ and $T = \emptyset$, then (G, δ) is a no-instance. If $G = K_2$ and $T = \emptyset$ then, trivially, $opt_S(G, \delta) = 0$, and if $G = K_1 + K_1$ and $T = V(G)$ then $opt_S(G, \delta) = 1$. To avoid these trivial instances, we therefore assume that G contains at least three vertices. Under these assumptions we will show that $opt_S(G, \delta)$ is always finite and give exact formulas for the value of $opt_S(G, \delta)$. Let p be the number of components of G that do not contain any vertex of T and let q be the number of components of G that contain at least one vertex of T. We prove the following series of statements:

- $opt_S(G, \delta) = 0$ if $p = 1, q = 0$,
- $opt_S(G, \delta) = \max\{3, p\}$ if $p \geq 2, q = 0$,
- $opt_S(G, \delta) = \frac{1}{2}|T| + 1$ if $p = 0, q = 1$, $G[T] = K_{1,r}$, for some $r \geq 1$, and each edge of $G[T]$ is a bridge of G,
- $opt_S(G, \delta) = \max\{p + q - 1, p + \frac{1}{2}|T|\}$ in all other cases.

Note that if $p = 1, q = 0$, then the first statement applies and the trivial solution $(A, D) = (\emptyset, \emptyset)$ is optimal. We now consider the remaining three cases separately.

Case 1: *$p \geq 2$ and $q = 0$.*
Then $T = \emptyset$, so by Lemma 3 for any semi-solution (A, D), every vertex in $G_S(A \cup D)$ must have even degree in $G_S(A \cup D)$. In other words, every vertex of G must be incident to an even number of edges in $A \cup D$. Since $p \geq 2$, the graph G is disconnected, so any solution (A, D) is non-empty. This means that $G_S(A \cup D)$ must contain a cycle, so $opt_S(G, \delta) \geq 3$ if a solution exits. Suppose $p = 2$. As G has at least three vertices, it contains a component containing an edge xy. Let z be a vertex in its other component. We set $A = \{xz, yz\}$ and $D = \{xy\}$ to obtain a solution for (G, δ). Since $|A| + |D| = 3$, this solution is optimal. Suppose $p \geq 3$. Since $G + A - D$ must be connected for any solution (A, D), every component in G must contain at least one vertex incident to an edge of A. By Lemma 3, this vertex must be incident to an even number of edges of $A \cup D$, meaning that it must be incident to at least two such edges. Therefore $opt_S(G, \delta) \geq p$. Indeed, if we choose vertices $v_1, \ldots, v_p$,

one from each component of G, then setting $A = \{v_1v_2, v_2v_3, \dots, v_{p-1}v_p, v_pv_1\}$ and $D = \emptyset$ gives a solution of size p, which is therefore optimal. This concludes Case 1.

Case 2: *$p = 0, q = 1$, $G[T] = K_{1,r}$ for some $r \geq 1$ and each edge of $G[T]$ is a bridge of G.*
Then G is connected. Let v_0 be the central vertex of the star and let $v_1, \dots, v_r$ be the leaves. By Lemma 3, in any semi-solution (A, D), every vertex of T must be incident to an odd number of edges in $A \cup D$, so $opt_S(G, \delta) \geq \frac{1}{2}|T|$. Suppose (A, D) is a semi-solution of size $|A| + |D| = \frac{1}{2}|T|$. Then $A \cup D$ must be a matching with each edge joining a pair of vertices of T. However, then $v_0v_i \in A \cup D$ for some i. Since $v_0v_i \in E(G)$, we must have $v_0v_i \in D$. However, since v_0v_i is a bridge of G, v_0 and v_i must then be in different components of $G + A - D$, so $G + A - D$ is not connected and (A, D) is not a solution. Therefore $opt_S(G, \delta) \geq \frac{1}{2}|T| + 1$.

Next we show how to find a solution of size $\frac{1}{2}|T| + 1$. Since $|T|$ is even, r must be odd. First suppose that $r = 1$. Since G is connected and v_0v_1 is a bridge, $G \setminus \{v_0v_1\}$ has exactly two components. Since G contains at least three vertices, one of these components contains another vertex x. Without loss of generality assume $xv_0 \in E(G)$, in which case $xv_1 \notin E(G)$. Then setting $A = \{xv_1\}$ and $D = \{xv_0\}$ gives a solution of size $|A|+|D| = 2 = \frac{1}{2}|T|+1$, so this solution is optimal. Now suppose $r \geq 3$. Let $A = \{v_1v_2, v_2v_3\} \cup \{v_{2i}v_{2i+1} \mid 2 \leq i \leq \frac{1}{2}(r - 1)\}$ and $D = \{v_0v_2\}$. Then (A, D) is a semi-solution and since $v_0, \dots, v_r$ are all in the same component of $G+A-D$, we find that (A, D) is a solution. Since $|A|+|D| = 2+\frac{1}{2}(r-1)-1+1 = \frac{1}{2}|T| + 1$, this solution is optimal. This concludes Case 2.

Case 3: *$q \geq 1$ and Case 2 does not hold.*
Then $T \neq \emptyset$. Let $G_1, \dots, G_p$ be the components of G without vertices of T and let $G' = G - V(G_1) \cup \dots \cup V(G_p)$. Note that $G' = G$ if $p = 0$ and that G' is not the empty graph, as $q > 0$. Choose $v_i \in V(G_i)$ for $i \in \{1, \dots, p\}$.

We first show that $opt_S(G, \delta) \geq \max\{p + q - 1, p + \frac{1}{2}|T|\}$. Since G has $p + q$ components, any solution (A, D) must contain at least $p + q - 1$ edges in A to ensure that $G + A - D$ is connected, so $opt_S(G, \delta) \geq p + q - 1$. If (A, D) is a solution then every component G_i must contain a vertex incident to some edge in A. By Lemma 3, this vertex must be incident to an even number of edges of $A \cup D$, meaning that it must be incident to at least two such edges. By Lemma 3, every vertex of T must be incident to some edge in $A \cup D$. Therefore $A \cup D$ must contain at least $p + \frac{1}{2}|T|$ edges, so $opt_S(G, \delta) \geq p + \frac{1}{2}|T|$.

We now show how to find a solution of size $\max\{p + q - 1, p + \frac{1}{2}|T|\}$. We start by finding a maximum matching M in $\overline{G[T]}$. Let U be the set of vertices in T that are not incident to any edge in M. We divide the argument into two cases, depending on the size of U.

Case 3a: *$U = \emptyset$.*
In this case, by Lemma 3, setting $A = M$ and $D = \emptyset$ gives a semi-solution. Now suppose that $uv, u'v' \in M$, such that uv is not a bridge in $G + M$ and the vertices u and u' are in different components of $G + M$. Let $M' = M \setminus \{uv, u'v'\} \cup \{u'v, uv'\}$. Then M' is also a maximum matching in $\overline{G[T]}$. However, $G + M'$ has one component less than $G + M$. Indeed, since uv is not a bridge in $G + M$, the vertices u, u', v, v' must all be in the same component of $G + M'$. Therefore, if such edges $uv, u'v' \in M$ exist, we replace M by M'. We do this exhaustively until no further such pairs of edges exist. At this point either every edge in M is a bridge in $G + M$ or every edge in M is in the same component of $G + M$. We consider these possibilities separately.

First suppose that every edge in M is a bridge in $G + M$. Choose $uv \in M$ and let $Q_1, \dots, Q_k$ be the components of $G + M$, with $u, v \in V(Q_1)$. Note that since every edge in M is a bridge, $k = p + q - |M|$. Now let $x_i \in V(Q_i)$ for $i \in \{2, \dots, k\}$. Let $D = \emptyset$

and let $A = M$ if $k = 1$ and $A = M \setminus \{uv\} \cup \{ux_2, x_2x_3, \ldots, x_{k-1}x_k, x_kv\}$ otherwise. Now every vertex in $G + A - D$ has the same degree parity as in $G + M$, so (A, D) is a semi-solution by Lemma 3. The graph $G + A - D$ is connected, so (A, D) is a solution. As $|A| + |D| = |M| - 1 + p + q - |M| + 0 = p + q - 1$, we find that (A, D) is an optimal solution.

Now suppose that every edge in M is in the same component of $G + M$. Note that $G_1, \ldots, G_p$ are the remaining components of $G + M$. Choose $uv \in M$. Let $D = \emptyset$ and let $A = M$ if $p = 0$ and $A = M \setminus \{uv\} \cup \{uv_1, v_1v_2, \ldots, v_{p-1}v_p, v_pv\}$ otherwise. Then every vertex in $G + A - D$ has the same parity as in $G + M$ and $G + A - D$ is connected, so by Lemma 3 (A, D) is a solution. Since $|A| + |D| = \frac{1}{2}|T| - 1 + p + 1 = p + \frac{1}{2}|T|$, this solution is optimal. This concludes Case 3a.

Case 3b: $U \neq \emptyset$.
Note that $z = |U|$ must be even since $|T|$ is even. Every pair of vertices in U must be non-adjacent in $\overline{G}$, as otherwise M would not be maximum. Therefore $G[U]$ is a clique. Let $U = \{u_1, \ldots, u_z\}$.

We claim that $Q = G' + M$ is connected. Clearly every vertex of the clique U must be in the same component of $Q = G' + M$. Suppose for contradiction that Q_1 is a component of Q that does not contain U. Then Q_1 must contain some edge $w_1w_2 \in M$. However, in this case $M' = M \setminus \{w_1w_2\} \cup \{u_1w_1, u_2w_2\}$ is a larger matching in $\overline{G[T]}$ than M, which contradicts the maximality of M. Therefore Q is connected.

Let $M' = \{u_1u_2, u_3u_4, \ldots, u_{z-1}u_z\}$. If $z \geq 4$ then since U is a clique, $G' + M - M'$ is connected. If $p = 0$ set $A = M$ and $D = M'$. If $p > 0$ set $A = M \cup \{u_1v_1, v_1v_2, \ldots, v_{p-1}v_p, v_pu_2\}$ and $D = M' \setminus \{u_1u_2\}$. Then $G + A - D$ is connected, so (A, D) is a solution by Lemma 3. This solution has size $|A| + |D| = p + \frac{1}{2}|T|$, so it is optimal.

Now suppose that $z \leq 3$. Then $z = 2$. If $p > 0$, let $A = M \cup \{u_1v_1, v_1v_2, \ldots, v_{p-1}v_p, v_pu_2\}$ and $D = \emptyset$. Then $G + A - D$ is connected, so (A, D) is a solution by Lemma 3. This solution has size $|A| + |D| = p + \frac{1}{2}|T|$, so it is optimal. Assume that $p = 0$, so $G + M$ contains only one component. If u_1u_2 is not a bridge in $G + M$, let $A = M$ and $D = \{u_1u_2\}$. Then $G + M$ is connected, so (A, D) is a solution. This solution has size $|A| + |D| = p + \frac{1}{2}|T|$, so it is optimal.

Now assume that u_1u_2 is a bridge in $Q = G + M$. Let Q_1 and Q_2 denote the components of $Q - \{u_1u_2\}$ with $u_1 \in V(Q_1)$ and $u_2 \in V(Q_2)$. Note that u_1u_2 is also a bridge in G. We claim that the edges of M are either all in Q_1 or all in Q_2. Suppose for contradiction that $y_1z_1 \in E(Q_1) \cap M$ and $y_2z_2 \in E(Q_2) \cap M$. Then $M' = M \setminus \{y_1z_1, y_2z_2\} \cup \{u_1y_2, u_2y_1, z_1z_2\}$ would be a larger matching in $\overline{G[T]}$ than M, contradicting the maximality of M. Without loss of generality, we may therefore assume that all edges of M are in Q_1.

Let $M = \{x_1y_1, \ldots, x_ry_r\}$, where $r = \frac{1}{2}|T| - 1$. We claim that u_1 must be adjacent in G to all vertices of $T \setminus \{u_1\}$. Suppose for contradiction that u_1 is non-adjacent in G to some vertex of $T \setminus \{u_1\}$. Since $u_1u_2 \in E(G)$, this vertex would have to be incident to some edge in M. Without loss of generality, assume $u_1x_1 \notin E(G)$. Then $M' = M \setminus \{x_1y_1\} \cup \{u_1x_1, u_2y_1\}$ would be a larger matching in $\overline{G[T]}$ than M, contradicting the maximality of M. Therefore u_1 is adjacent in G to every vertex of $T \setminus \{u_1\}$. In particular, since $p = 0$, it follows that $q = 1$ and G is connected.

Suppose that every edge between u_1 and $T \setminus \{u_1\}$ is a bridge in G. Then no two vertices of $T \setminus \{u_1\}$ can be adjacent, and $G[T] = K_{1,r}$. However, then Case 2 applies, which we assumed was not the case. Without loss of generality, we may therefore assume that u_1x_1 is not a bridge in G. Let $A = M \setminus \{x_1y_1\} \cup \{y_1u_2\}$ and $D = \{u_1x_1\}$. Then $G + A - D$ is connected, so (A, D) is a solution. Since $|A| + |D| = \frac{1}{2}|T| - 1 - 1 + 1 + 1 = p + \frac{1}{2}|T|$, this solution is optimal. This concludes Case 3b and therefore also concludes Case 3.

It is clear that $opt_S(G, \delta)$ can be computed in $O(n+m)$ time. We also observe that the above proof is constructive, that is, we not only solve the decision variant of CDPE(ea, ed) but we can also find an optimal solution. To do so, we must find a maximum matching in $\overline{G[T]}$. This takes $O(n^{5/2})$ time [18]. However, the bottleneck is in Case 3a, where we are glueing components by replacing two matching edges by two other matching edges, which takes $O(n^2)$ time. As the total number of times we may need to do this is $O(n)$, this procedure may take $O(n^3)$ time in total. Hence, we can obtain an optimal solution in $O(n^3)$ time. ◀

The proof of the next result has been omitted.

▶ **Theorem 6.** *Let* $\{vd\} \subseteq S \subseteq \{vd, ed, ea\}$*. Then* CDPE$(S)$ *is* NP*-complete and* W[1]*-hard when parameterized by* k*, even if* $\delta \equiv 0$.

4 Connected Degree Balance Editing

We will show that CDBE(S) is polynomial-time solvable if $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$ and that it is NP-complete and W[1]-hard with parameter k if $\mathsf{vd} \in S$.

Let $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$. Let (G, δ, k) be an instance of CDBE(S) with $G = (V, E)$. Let A be a set of arcs not in G, and let D be a set of arcs in G, with $D = \emptyset$ if $S = \{\mathsf{ea}\}$. We say that (A, D) is a *solution* for (G, δ, k) if its *size* $|A| + |D| \leq k$, the equation $d^+_H(u) - d^-_H(u) = \delta(u)$ holds for every vertex u and the graph $H = G + A - D$ is connected; if H is not connected then (A, D) is a *semi-solution* for (G, δ, k). Just as in Section 3 we consider the optimization version for CDBE(S) and we use the same terminology.

Let (G, δ) be an instance of (the optimization version) of CDBE(S) where $G = (V, E)$. Let $T = T_{(G,\delta)}$ be the set of vertices v such that $d^+_G(v) - d^-_G(v) \neq \delta(v)$. Define a function $f_{(G,\delta)} : T \to \mathbb{Z}$ by $f(v) = f_{(G,\delta)}(v) = \delta(v) - d^+_G(v) + d^-_G(v)$ for every $v \in T$.

We construct a directed multigraph G_S with vertex set V and arc set determined as follows. If $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$, for each pair of distinct vertices u and v in G, if $(u, v) \notin E$, add the arc (u, v) to G_S (these arcs are precisely those that can be added to G). If $S = \{\mathsf{ea}, \mathsf{ed}\}$, for each pair of distinct vertices u and v, if $(u, v) \in E$, add the arc (v, u) to G_S (these arcs are precisely those whose reverse can be deleted from G). Note that adding a (missing) arc has the same effect on the degree balance of the vertices in a digraph as deleting the reverse of the arc (if it exists). Also observe that G_S becomes a directed multigraph rather than a digraph only if $S = \{\mathsf{ea}, \mathsf{ed}\}$ and there are distinct vertices u and v such that $(u, v) \in E$ and $(v, u) \notin E$ applies. Moreover, G_S contains at most two copies of any arc, and if there are two copies of (u, v) then (v, u) is not in G_S.

Let F be a minimum directed f-join in G_S (if one exists). Note that F may contains two copies of the same arc if G_S is a directed multigraph. Also note that for any pair of vertices u, v, either $(u, v) \notin F$ or $(v, u) \notin F$, otherwise $F' = F \setminus \{(u, v), (v, u)\}$ would be a smaller f-join in G_S, contradicting the minimality of F. We define two sets A_F and D_F which, as we will show, correspond to a semi-solution (A_F, D_F) of (G, δ). Initially set $A_F = D_F = \emptyset$. Consider the arcs in F. If F contains (u, v) exactly once then add (u, v) to A_F if $(u, v) \notin E$ and add (v, u) to D_F if $(u, v) \in E$ (in this case $(v, u) \in E$ holds). If F contains two copies of (u, v) then add (u, v) to A_F and (v, u) to D_F; note that by definition of F and G_S, in this case $S = \{\mathsf{ea}, \mathsf{ed}\}, (u, v) \notin E$ and $(v, u) \in E$. Observe that the sets A_F and D_F are not multisets.

If X and Y are sets, then $X \uplus Y$ is the multiset that consists of one copy of each element that occurs in exactly one of X and Y and two copies of each element that occurs in both. The next lemma provides the starting point for our algorithm. Its proof has been omitted.

▶ **Lemma 7.** *Let $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$. Let (G, δ) be an instance of* $\mathrm{CDBE}(S)$ *where $G = (V, E)$. The following holds:*

(i) *If F is a minimum directed f-join in G_S, then (A_F, D_F) is a semi-solution for (G, δ) of size $|F|$.*

(ii) *If (A, D) is a semi-solution for (G, δ), then $A \uplus D^R$ is a directed f-join in G_S of size $|A| + |D|$.*

Let (G, δ) be an instance of $\mathrm{CDBE}(S)$. Let $p = p_{(G,\delta)}$ be the number of components of G that contain no vertex of T. Let $q = q_{(G,\delta)}$ be the number of components of G that contain at least one vertex of T. Let $t = t_{(G,\delta)} = \sum_{u \in T} |f(u)|$.

We now state the following lemma. Its proof (based on Lemmas 2 and 7) has been omitted.

▶ **Lemma 8.** *Let $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$. Let (G, δ) be an instance of* $\mathrm{CDBE}(S)$ *with $q \geq 1$. If F is a (given) minimum directed f-join in G_S, then (G, δ) has a solution that has size at most* $\max\{|F|, p + q - 1, p + \frac{1}{2}t\}$*, which can be found in $O(nm)$ time.*

The next result is our first main result of this section. We prove it by showing that the upper bound in Lemma 8 is also a lower bound for (almost) any instance of $\mathrm{CDBE}(S)$ with $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$ that has a semi-solution.

▶ **Theorem 9.** *For $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$,* $\mathrm{CDBE}(S)$ *can be solved in time $O(n^3 \log n \log\log n)$.*

Proof. Let $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$, and let (G, δ) be an instance of $\mathrm{CDBE}(S)$. We first use Lemma 2 to check whether G_S has a directed f-join. Because G_S has at most $2n^2$ arcs, this takes $O(n^3 \log n \log\log n)$ time. If G_S has no directed f-join then (G, δ) has no semi-solution by Lemma 7, and thus no solution either. Assume that G_S has a directed f-join, and let F be a minimum directed f-join that can be found in time $O(n^3 \log n \log\log n)$ by Lemma 2. As before, p denotes the number of components of G that do not contain any vertex of T, while q is the number of components of G that contain at least one vertex of T, and $t = \sum_{u \in T} |f(u)|$.

We will prove the following series of statements:

- $opt_S(G, \delta) = 0$ if $p \leq 1$, $q = 0$,
- $opt_S(G, \delta) = p$ if $p \geq 2$, $q = 0$,
- $opt_S(G, \delta) = \max(|F|, p + q - 1, p + \frac{1}{2}t)$ if $q > 0$.

If $p \leq 1$ and $q = 0$ then $A = D = \emptyset$ is an optimal solution. If $p \geq 2$ and $q = 0$, to ensure connectivity and preserve degree balance, for every component of G there must be at least one arc whose head is in this component and at least one arc whose tail is in this component, thus any solution must contain at least p arcs. Let $G_1, \ldots, G_p$ be the components of G and arbitrarily choose vertices $v_i \in V(G_i)$ for $i \in \{1, \ldots, p\}$. Let $A = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{p-1}, v_p), (v_p, v_1)\}$ and $D = \emptyset$. Then (A, D) is a solution which has size p and is therefore optimal.

Suppose $q \geq 1$. By Lemma 8 we find a solution (A, D) for (G, δ) of size at most $\max\{|F|, p + q - 1, p + \frac{1}{2}t\}$ in $O(nm)$ time. Hence, the total running time is $O(n^3 \log n \log\log n)$, and it remains to show that any solution has size at least $\max(|F|, p + q - 1, p + \frac{1}{2}t)$.

Let (A, D) be an arbitrary solution. Then (A, D) is also semi-solution. Every semi-solution has size at least $|F|$ by Lemma 7 2. Therefore (A, D) has size at least $|F|$.

Since there are $p + q$ components in G, we must add at least $p + q - 1$ arcs to ensure $G + A - D$ is connected. Therefore (A, D) has size at least $p + q - 1$.

Finally, for every vertex u with $f(u) > 0$ (resp. $f(u) < 0$) we find that (A, D) must be such that at least $|f(u)|$ arcs are either in A and have u as a tail (resp. head) or else are in D and have u as a head (resp. tail). For every component containing only vertices v with $f(v) = 0$, there must be at least one arc in A whose head is in this component and at least one arc in A whose tail is in this component (to ensure connectivity and to ensure that the degree balance is not changed for any vertex in this component). Therefore we have that (A, D) has size at least $p + \frac{1}{2}t$. This completes the proof of Theorem 9. ◀

The proof of our second main result of this section has been omitted.

▶ **Theorem 10.** *Let* $\{\mathsf{vd}\} \subseteq S \subseteq \{\mathsf{vd}, \mathsf{ed}, \mathsf{ea}\}$*. Then* CDBE$(S)$ *is* NP*-complete and* W[1]*-hard when parameterized by* k*, even if* $\delta \equiv 0$*.*

5 Conclusions

By extending previous work [1, 4, 7] we completely classified both the classical and parameterized complexity of CDPE(S) and CDBE(S), as summarized in Table 1. Our work followed the framework used [11, 17] for (Connected) Degree Constraint Editing(S). Our study was motivated by Eulerian graphs. As such, the variants DPE(S) and DBE(S) of CDPE(S) and CDBE(S), respectively, in which the graph H is no longer required to be connected, were beyond the scope of this paper. It follows from results of Cai and Yang [4] and Cygan [7], respectively, that for $S = \{\mathsf{vd}\}$, DPE(S) and DBE(S) are NP-complete and, when parameterized by k, W[1]-hard, whereas they are polynomial-time solvable for $S = \{\mathsf{ed}\}$ as a result of Lemmas 1 and 2, respectively. The problems DPE(S) and DBE(S) are also polynomial-time solvable if $\{\mathsf{ea}\} \subseteq S \subseteq \{\mathsf{ea}, \mathsf{ed}\}$; this is in fact proven by combining Lemmas 1 and 3 for the undirected case, and Lemmas 2 and 7 for the directed case. We expect the remaining (hardness) results of Table 1 to carry over as well.

Let ℓ be an integer. Here is a natural generalization of CDPE(S).

ℓ-CDME(S):	Connected Degree Modulo-ℓ-Editing(S)
Instance:	A graph G, integer k and a function $\delta\colon V(G) \to \{0, \dots, \ell - 1\}$.
Question:	Can G be (S, k)-modified into a connected graph H with $d_H(v) \equiv \delta(v) \pmod{\ell}$ for each $v \in V(H)$?

Note that 2-CDME(S) is CDPE(S). The following theorem shows that the complexity of 3-CDME(S) may differ from 2-CDME(S).

▶ **Theorem 11.** 3-CDME$(\{\mathsf{ea}, \mathsf{ed}\})$ *is* NP*-complete even if* $\delta \equiv 2$*.*

Proof. Reduce from the Hamiltonicity problem, which is NP-complete for connected cubic graphs [10]. Let G be a connected cubic graph. Let $\delta(v) = 2$ for every $v \in V(G)$, and take $k = |E(G)| - |V(G)|$. Then G has a Hamiltonian cycle if and only if G can be (S, k)-modified into a connected graph H with $d_H(v) = 2 \pmod 3$ for all $v \in V(H)$. ◀

It is natural to ask whether 3-CDME$(\{\mathsf{ea}, \mathsf{ed}\})$ is fixed-parameter tractable with parameter k.

Finally, another direction for future research is to investigate how the complexity of CDPE(S) and CDBE(S) changes if we permit other graph operations, such as edge contraction, to be in the set S.

References

1 Francis T. Boesch, Charles L. Suffel, and Ralph Tindell. The spanning subgraphs of Eulerian graphs. *Journal of Graph Theory*, 1(1):79–84, 1977.

2 Pablo Burzyn, Flavia Bonomo, and Guillermo Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006.

3 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

4 Leizhen Cai and Boting Yang. Parameterized complexity of even/odd subgraph problems. *Journal of Discrete Algorithms*, 9(3):231–240, 2011.

5 Katarína Cechlárová and Ildikó Schlotter. Computing the deficiency of housing markets with duplicate houses. In *IPEC 2010*, volume 6478 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2010.

6 Robert Crowston, Gregory Gutin, Mark Jones, and Anders Yeo. Parameterized Eulerian strong component arc deletion problem on tournaments. *Information Processing Letters*, 112(6):249–251, 2012.

7 Marek Cygan, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Ildikó Schlotter. Parameterized complexity of Eulerian deletion problems. *Algorithmica*, 68(1):41–61, 2014.

8 Frederic Dorn, Hannes Moser, Rolf Niedermeier, and Mathias Weller. Efficient algorithms for Eulerian extension and rural postman. *SIAM Journal on Discrete Mathematics*, 27:75–94, 2013.

9 Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

10 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

11 Petr A. Golovach. Editing to a connected graph of given degrees. In *MFCS Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 2014.

12 Petr A. Golovach. Editing to a graph of given degrees. In *IPEC 2014*, *Lecture Notes in Computer Science*. Springer, to appear.

13 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

14 Wiebke Höhn, Tobias Jacobs, and Nicole Megow. On Eulerian extensions and their application to no-wait flowshop scheduling. *Journal of Scheduling*, 15(3):295–309, 2012.

15 Linda Lesniak and Ortrud R. Oellermann. An Eulerian exposition. *Journal of Graph Theory*, 10(3):277–297, 1986.

16 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

17 Luke Mathieson and Stefan Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *Journal of Computer and System Sciences*, 78(1):179–191, 2012.

18 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *FOCS 1980*, pages 17–27. IEEE Computer Society, 1980.

19 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009.

20 Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001.

Parameterized Complexity of Fixed-Variable Logics

Christoph Berkholz[1] **and Michael Elberfeld**[2]

1 **RWTH Aachen University, Aachen, Germany**
`berkholz@cs.rwth-aachen.de`
2 **RWTH Aachen University, Aachen, Germany**
`elberfeld@cs.rwth-aachen.de`

Abstract

We study the complexity of model checking formulas in first-order logic parameterized by the number of distinct variables in the formula. This problem, which is not known to be fixed-parameter tractable, resisted to be properly classified in the context of parameterized complexity. We show that it is complete for a newly-defined complexity class that we propose as an analog of the classical class PSPACE in parameterized complexity. We support this intuition by the following findings: First, the proposed class admits a definition in terms of alternating Turing machines in a similar way as PSPACE can be defined in terms of polynomial-time alternating machines. Second, we show that parameterized versions of other PSPACE-complete problems, like winning certain pebble games and finding restricted resolution refutations, are complete for this class.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Parameterized complexity, polynomial space, first-order logic, pebble games, regular resolutions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.109

1 Introduction

The main goal of computational complexity theory is to distinguish between tractable and intractable problems. In classical complexity theory tractable problems are those that can be solved in polynomial time, whereas intractable problems require exponential time (most notably NP-complete problems, but also problems complete for higher levels of the polynomial hierarchy, PSPACE, and EXPTIME). In parameterized complexity theory, tractable problems are in FPT and can be solved in time $f(k) \cdot n^{O(1)}$, whereas intractable problems require a running time of $n^{f(k)}$.

Beside distinguishing between just tractable and intractable problems, looking at different levels of intractability for NP-hard problems (by comparing them with respect to polynomial-time reductions) led to understanding the importance of the polynomial hierarchy as well as polynomial-space, and exponential-time computations. Already during the incubation of parameterized complexity theory different levels of parameterized intractability were observed based on comparing problems with respect to fixed-parameter tractable reduction (fpt-reductions). This turned into a flourishing research area where classes that were initially defined in an adhoc way by considering yet unclassified problems and their closures under fpt-reductions turned out to be definable using descriptive characterizations in terms of first-order logic.

Paper's Issue. The class XP of parameterized problems that can be solved in time $n^{f(k)}$ is the analog of EXPTIME in parameterized complexity as both classes contain all problems of intractable running time. Indeed, it turned out that parameterizations of EXPTIME-complete problems tend to be complete for XP [2, 3, 4]. Several classes are proposed as parameterized analogs of PSPACE. This includes the classes AW[SAT] and AW[*], which both admit alternative characterizations in terms of model checking first-order formulas, as well as AW[P], which can be defined based on deterministic Turing machines that access a certificate containing blocks of existential (nondeterministic) bits and blocks of universal (nondeterministic) bits. For all of these PSPACE-analogs the alternation used to solve problems is bounded by a function in terms of the parameter. On the one side, the classes defined in this way helped to classify parameterized versions of PSPACE-complete problems where the parameter is used to bound the use of alternation. On the other side, problems that result from more general parameterizations of PSPACE-complete problems resisted to be classified using these classes. A prominent example is the problem of evaluating first-order formulas. It is PSPACE-complete [18] in the classical setting, and known to be in XP when parameterized by the number of distinct variables in the formula, but not known to be hard for this class. The importance of the fixed variable fragments of first-order logic stems from the fact that k-variable formulas can be evaluated in time $n^{O(k)}$. In addition, by reusing the variables one has access to an unlimited number of quantifier alternations, which makes this fragment much more expressive than fragments with bound quantifier depth. A similar observation can be made for determining the winner in a classical acyclic pebble game and finding linear depth resolution refutations of bounded width; their unparameterized versions are shown to be complete for PSPACE in [14] and [3], respectively, but they are not complete for known parameterized analogs of PSPACE as they require unbounded alternation.

Paper's Contributions. We properly classify the parameterized complexity of these problems by presenting the following contributions during the course of the present paper: (1) We consider the closure under fpt-reductions of model checking first-order formulas parameterized by the number of distinct variables in the formula and sort this class into the hierarchy of known levels of parameterized intractability. (2) We prove that the newly defined class SXP, which stands for shallow XP, has a natural characterization in terms of alternating Turing machines (with unbounded alternation) in a similar way as PSPACE can be characterized in terms of alternating polynomial time. For this result, we apply techniques from descriptive complexity theory [13] to simulate the behavior of alternating machines using first-order formulas. (3) We show that other PSPACE-complete problems are complete for this class under fpt-reductions when parameterized in a natural and very general way. We first simulate the model checking game for k-variable logic within the acyclic $k+2$-pebble game of Kasai, Adachi and Iwata, which was introduced to simulate PSPACE machines, to show that the pebble game is complete for our new class when parameterized by the number of pebbles. Afterwards, we use a known reduction from the acyclic pebble game to regular resolution of bounded width to show that finding resolution refutations of linear depth and width k is another PSPACE-complete problem that fits in our parameterized analog of PSPACE when parameterized by the width. Interestingly, the pebble game and bounded width resolution have more general versions that are EXPTIME-complete, classically, and XP-complete, parameterized.

Paper's Organization. The next section defines concepts and terminology related to parameterized complexity and first-order logic. The subsequent Sections 3, 4, 5 present, respectively, the definition of our newly proposed parameterized analog of PSPACE, a machine characterization for the class, and complete problems.

2 Background

The present section provides background from parameterized complexity and first-order logic as well as establishes notation related to parameterized versions of model checking first-order formulas. The used definitions and notations closely follow the book of Flum and Grohe [12]; see also this book for standard results in parameterized complexity mentioned below.

Parameterized complexity. A *parameterized problem* is a pair (P, κ) consisting of a *(classical) problem* $P \subseteq \{0,1\}^*$ and a *parameterization* $\kappa\colon \{0,1\}^* \to \{1\}^*$ that is polynomial-time computable; we commonly denote it by p-κ-P. Given an *instance* $x \in \{0,1\}^*$, we use the shorthands $n := |x|$, its *size*, and $k := |\kappa(x)|$, its *parameter*. We denote by FPT the class of parameterized problems (P, κ) that are solvable by a deterministic Turing machine (DTM) whose runtime is at most $f(k) \cdot n^{O(1)}$ for a computable function $f\colon \mathbb{N} \to \mathbb{N}$; (parameterized) problems in FPT are *fixed-parameter tractable*. The class XP is defined like FPT, but using time bounds $n^{f(k)}$. The deterministic time hierarchy theorem implies that FPT is a proper subclass of XP.

An fpt-*reduction* from a parameterized problem (P,κ) to a parameterized problem (P',κ') is a mapping $r\colon \{0,1\}^* \to \{0,1\}^*$ that is computable by a DTM in time $f(k) \cdot n^{O(1)}$ for a computable function $f\colon \mathbb{N} \to \mathbb{N}$, such that for every $x \in \{0,1\}^*$, we have (1) $x \in P$ if, and only if, $r(x) \in P'$, and (2) $|\kappa'(r(x))| \leq |g(\kappa(x))|$ for some reduction-dependent function $g\colon \{1\}^* \to \{1\}^*$. Given a parameterized problem (P', κ'), the *closure of* (P', κ') *under* fpt-*reductions*, denoted by $[(P', \kappa')]^{\text{fpt}}$, is the class of all problems (P, κ) with an fpt-reduction from (P, κ) to (P', κ'). Later we study problems that are complete for XP or other complexity classes of parameterized problems between FPT and XP. In all of these cases, completeness is defined with respect to fpt-reductions.

First-order logic. We start to define the syntax and semantics of first-order logic: A *vocabulary* τ is a nonempty and finite set of *relation symbols* R_i with *arities* $\text{arity}(R_i) \in \mathbb{N}$. A *structure* $\mathcal{A}$ over τ consists of a finite set A, its *universe*, and a *relation* $R_i^{\mathcal{A}} \subseteq A^{\text{arity}(R_i)}$ for every relation symbol R_i of τ. Based on *(element) variables* x_i, $i \in \mathbb{N}$, *first-order formulas* (FO-formulas) over a vocabulary τ are (1) *atomic* formulas $x_i = x_j$ and $(x_1, \dots, x_r) \in R_i$, and (2) *composed* formulas $\neg\varphi$, $\varphi \wedge \psi$ and $\varphi \vee \psi$, which are based on *connectives*, and $\exists x_i\, \varphi$ and $\forall x_i\, \varphi$, which are based on *quantifiers*. Given a structure $\mathcal{A}$ and a formula φ over the same vocabulary τ, $\mathcal{A}$ *satisfies* φ if the usual model relation $\mathcal{A} \models \varphi$ holds. Two formulas φ and ψ are *equivalent* if they are satisfied by exactly the same structures.

In order to define parameterized problems and complexity classes that are based on first-order formulas, we define classes of formulas and parameters of formulas: First of all, we only consider FO-formulas in *negation normal form*, whose $\neg$-connectives are immediately in front of atomic formulas. This does not restrict the set of FO-formulas in the sense that every FO-formula can be turned into an equivalent formula in negation normal form by recursively applying the rules of De Morgan. We denote this set of formulas by FO. The *quantifier alternation depth* of a formula φ, denoted by $\text{qad}(\varphi)$, is the number of alternations from $\exists$- to $\forall$-quantifiers on any root-to-leaf path in φ's syntax tree plus 1; and plus 2 if the first quantifier is $\forall$. For every $t \in \mathbb{N}$, the class of formulas $\varphi \in$ FO with $\text{qad}(\varphi) = t$ is ALT_t. A formula φ is in *prenex normal form* if $\varphi = Q_1 x_1 \dots Q_\ell x_\ell\, \psi$ where the Q_i are quantifiers and

ψ does not contain quantifiers. The class of all these formulas is $\textsc{prenex}$ and, for every $t \in \mathbb{N}$, we set $\Sigma_t := \textsc{prenex} \cap \textsc{alt}_t$. Let $\varphi \in \textsc{fo}$. We denote the number of distinct variables of φ by $\mathrm{var}(\varphi)$. The size of φ, denoted by $\mathrm{size}(\varphi)$, is the total number of symbols used to write down φ.

Model checking formulas. In order to consider vocabularies, structures, and formulas as part of instances to computational problems, we encode them using binary strings in a standard way as done in [12]. We write $\mathrm{enc}(\tau)$ for the binary string encoding of a vocabulary τ; $\mathrm{enc}(\mathcal{A})$ and $\mathrm{enc}(\varphi)$ are used for the encoding of a structure $\mathcal{A}$ and a formula φ, respectively. Given a class of first-order formulas $\mathcal{C} \subseteq \textsc{fo}$, we denote by $\textsc{mc}(C)$ the *model checking problem* for formulas from C: A positive instance of $\textsc{mc}(C)$ consists of an encoded vocabulary $\mathrm{enc}(\tau)$ as well as an encoded structure $\mathrm{enc}(\mathcal{A})$ and an encoded formula $\mathrm{enc}(\varphi)$ with (1) $\varphi \in C$, (2) $\mathcal{A}$ and φ are both over τ, and (3) $\mathcal{A} \models \varphi$. We consider parameterized versions of model checking problems with respect to several classes of formulas in the following section, and their closure under fpt-reductions.

3 Parameterized complexity of first-order logic

Previous studies (mainly subsumed by the book [12]) observed that the parameterized complexity of model checking first-order formulas with respect to several classes of formulas and parameterizations is not only interesting from the perspective of solving the problem, but as a central concept to study parameterized intractability, itself. That means, model checking formulas in $\textsc{fo}$-logic is commonly used to define levels of parameterized intractability, which make up a candidate hierarchy of complexity classes between the, provably distinct, classes FPT and XP.

A first family of classes are defined as *analogs of* NP in the parameterized setting: This spans the W-hierarchy, with its frequently used first level $\mathrm{W}[1] := [\text{p-size-}\textsc{mc}(\Sigma_1)]^{\mathrm{fpt}}$ [7, 9, 11], as well as $\mathrm{W}[\mathrm{SAT}] := [\text{p-var-}\textsc{mc}(\Sigma_1)]^{\mathrm{fpt}}$ [7, 16] and the class W[P], whose definition equals the one of FPT except that the used deterministic Turing machines have access to a nondeterministic certificate of $f(k) \cdot \log n$ nondeterministic bits [7, 6]. These classes are considered to be analogs of NP in the parameterized world since it is possible to nondeterministically guess candidate solutions to, say, graph problems of a parameter-bounded size, and verify the guessed solution. The computational power we have for the verifying step depends on the particular class. Since we consider FPT as the lowest level, of tractable problems, the classes are defined by taking the closure of these problems under fpt-reductions.

In a similar fashion, parameterized *analogs of the polynomial-time hierarchy (*PH*)* are defined based on formulas with constant alternations with the most prominent suggestion being the A-hierarchy with levels $A[t] := [\text{p-size-}\textsc{mc}(\Sigma_t)]^{\mathrm{fpt}}$ for $t \in \mathbb{N}$ [10, 11].

A third kind of classes are designed to be *analogs of* PSPACE in the parameterized setting: In this case, not only existential, but also universal, nondeterminism is permitted; still nondeterminism is bounded in terms of the parameter. The most powerful of these classes is AW[P], which is defined as W[P], but this time the acceptance behavior of the $\textsc{dtm}$ depends on a length-$(f(k) \cdot \log n)$ certificate containing both existential and universal nondeterministic bits where the number of alternations is bounded by $f(k)$. Classes that are defined via a less powerful solution verification phase are $\mathrm{AW}[\mathrm{SAT}] := [\text{p-var-}\textsc{mc}(\textsc{prenex})]^{\mathrm{fpt}}$ and $\mathrm{AW}[*] := [\text{p-size-}\textsc{mc}(\textsc{prenex})]^{\mathrm{fpt}}$. All of the mentioned PSPACE-analogs are originally defined in terms of Boolean circuits and propositional logic [1]; the definitions based on machines and model checking first-order formulas are taken from [12, Chapter 8].

While the parameterized approaches towards mirroring the behavior of PSPACE have proven to be useful to classify a large number of problems, some problems remained unclassified and, hence, not well understood. These are parameterized problems whose solutions are based on existential and universally nondeterministic guesses, but where it seems not possible to bound the nondeterministic guesses in terms of the parameter. Thus, the parameter seems to play a different role for these problems. A prominent example of such a problem is model checking first-order formulas that only use a fixed (parameter-bounded) number of distinct variables. As the variables can be reused, the quantifier alternation does not depend on the parameter. This observation leads us to defining the following *parameterized analog* of PSPACE with unbounded alternations. In Section 4 we realize that the defined complexity class has a characterization in terms of alternating Turing machines that is similar to XP, but using shallow parallel computations; hence, we choose the name SXP with S standing for "shallow".

▶ **Definition 1.** $\mathrm{SXP} := [\text{p-var-}\textsc{mc}(\textsc{fo})]^{\mathrm{fpt}}$

The class SXP is contained in XP and contains AW[P]. Figure 1 shows the relations between these and other classes from parameterized complexity mentioned above.

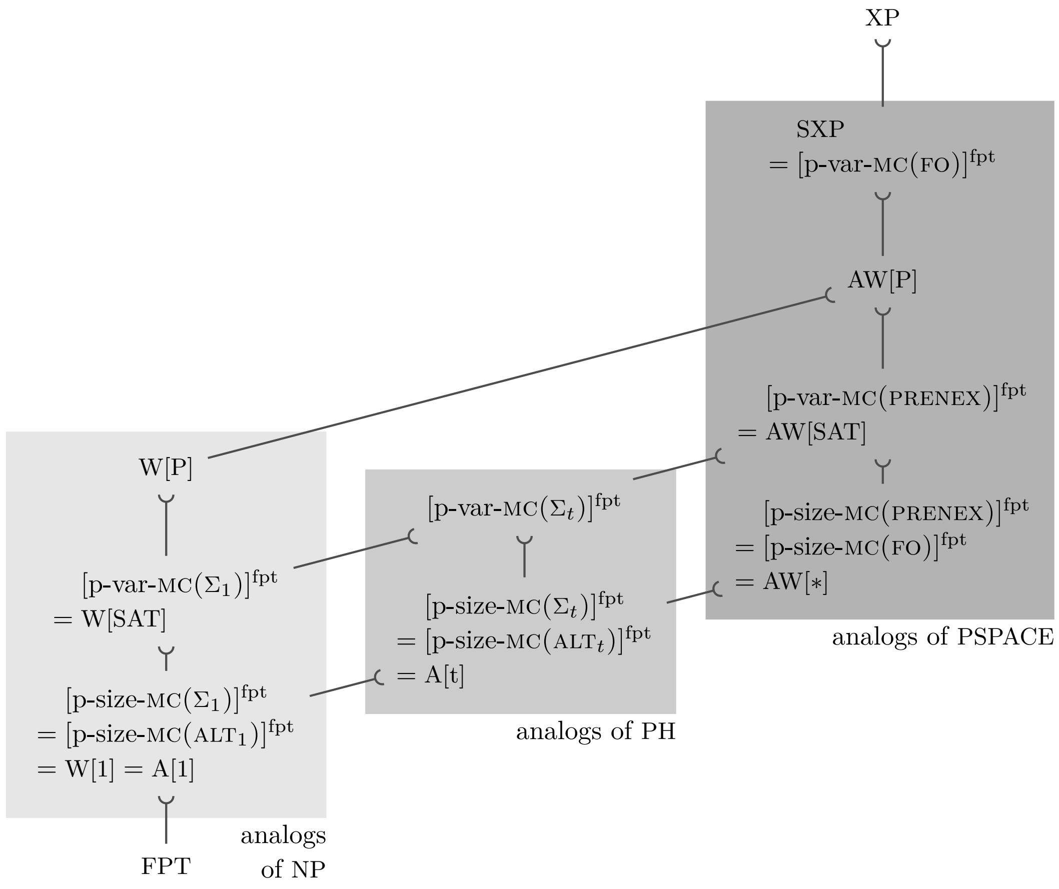
▶ **Fact 2.** $\mathrm{SXP} \subseteq \mathrm{XP}$

▶ **Lemma 3.** $\mathrm{AW[P]} \subseteq \mathrm{SXP}$

Proof Sketch. Let $(P, \kappa) \in \mathrm{AW[P]}$ via a DTM M running in time $f(k) \cdot n^c$ for some computable function f and constant c, and using nondeterministic certificates of length $f(k) \cdot \log n$ of existential and universal bits; with $f(k)$ alternations. The first phase of our fpt-reduction to p-var-MC(FO) reduces the problem of simulating the computation of M on length-n and parameter-k instances to the circuit evaluation problem as described in [15] for the classical class P. The second step replaces the task of evaluating the circuit by model checking a constant-variable first-order formula that defines the evaluation problem for circuits (the same approach is commonly used to show that model checking first-order formulas with just 2 variables is complete for P). To also take the alternating certificate into account, which is given to the DTM, our formula is enriched by existential quantifiers, which guess existential bits, and universal quantifiers, which guess universal bits; this construction uses a single quantifier to handle a length-$(\log n)$ substring of the certificate by first applying the "$k \cdot \log n$"-trick (see [12, Corollary 3.13] for details) to the circuit. Both the circuit and the formula can be constructed using an fpt-reduction. The number of variables used by the formula is bounded by a function in the original parameter. ◀

4 Alternative Characterizations

The classes that are defined via model checking first-order formulas in the previous section are all defined by first taking a prototypical model checking problem for first-order formulas of a restricted syntax along with a parameter, which is the formulas size or number of distinct variables. Then the closures of these problems under fpt-reductions are considered, which captures certain kinds of parameterized intractable problems. The prototypical problems are chosen in order to mirror the behavior of a classical class in the parameterized setting. In this section we present an alternative characterization of SXP in terms of alternating Turing machines. It shows how the behavior of the class SXP (of parameterized problems) parallels the behavior of the class PSPACE (of classical problems).



XP
SXP
= [p-var-MC(FO)]$^{\text{fpt}}$
AW[P]
[p-var-MC(PRENEX)]$^{\text{fpt}}$
= AW[SAT]
[p-size-MC(PRENEX)]$^{\text{fpt}}$
= [p-size-MC(FO)]$^{\text{fpt}}$
= AW[*]
analogs of PSPACE
[p-var-MC(Σ_t)]$^{\text{fpt}}$
[p-size-MC(Σ_t)]$^{\text{fpt}}$
= [p-size-MC(ALT$_t$)]$^{\text{fpt}}$
= A[t]
analogs of PH
W[P]
[p-var-MC(Σ_1)]$^{\text{fpt}}$
= W[SAT]
[p-size-MC(Σ_1)]$^{\text{fpt}}$
= [p-size-MC(ALT$_1$)]$^{\text{fpt}}$
= W[1] = A[1]
FPT
analogs of NP

Figure 1 We have the following inclusions of parameterized complexity classes for every $t \in \mathbb{N}$; where C —⊂ D indicates C $\subseteq$ D for classes C and D. While FPT is commonly considered to be the analog of P in parameterized complexity and XP is an analog of EXPTIME, there are several suggestion to reflect the behavior of the classical classes NP, the levels of PH, and PSPACE. Our suggestion for a parameterized version of PSPACE is based on parameterizing first-order model checking via the number of distinct variables in formulas.

An *alternating Turing machine* (ATM) M consists of a set of *states* Q that is partitioned into a set of *existential states* $Q_\exists$ and a set of *universal states* $Q_\forall$. Its *(nondeterministic) transitions* are encoded by a relation $\Delta \subseteq Q \times \Sigma^k \times Q \times \Sigma^k \times \{\texttt{LEFT}, \texttt{RIGHT}\}^k$ where $(q, \sigma_1, \dots, \sigma_k, q', \sigma'_1, \dots, \sigma'_k, d_1, \dots, d_k) \in \Delta$ means that if M is in state q and reads the symbol σ_i on tape i, for $i \in \{1, \dots, k\}$, then it can write the symbol σ'_i on tape i, for $i \in \{1, \dots, k\}$, and moves its heads as defined by the d_i. We consider only machines M that halt on every computation path. The acceptance behavior of an ATM is defined recursively (without using accepting and rejecting states explicitly) as follows: A *universal* configuration *accepts* if every immediate successor configuration accepts, an *existential* configuration *accepts* if there exists an immediate successor configuration that accepts. M accepts an input if the *starting configuration* accepts.

From the proof of the well known characterization from Chandra et al. [5] of polynomial deterministic time in terms of alternating logarithmic space, we get the following alternative definition of XP. This parallels the definition of EXPTIME in terms of ATMs using polynomial space.

▶ **Fact 4.** XP *is the class of parameterized problems* (P, κ) *that are accepted by* ATMs *using space at most* $f(k) \cdot \log n$.

While for EXPTIME problems we do not hope to lower the run-time substantially by using alternation, alternation speeds-up the solution of problems in PSPACE since it equals the class of problems accepted by ATMs in polynomial time [5]. The following lemma states that our proposed parameterized version of PSPACE has a similar behavior. Its problems can be solved by ATMs using $f(k) \cdot \log n$ space, but only running in $f(k) \cdot n^{O(1)}$ time. The proof of the lemma is based on a refined view on the PSPACE-completeness of model checking first-order formulas [17, 18] as well as ideas from descriptive complexity theory [13].

▶ **Theorem 5.** SXP *is the class of parameterized problems* (P, κ) *that are accepted by* ATMs *using space at most* $f(k) \cdot \log n$ *and running in time at most* $f(k) \cdot n^{O(1)}$.

Proof. We start to show how problems in SXP, which are fpt-reducible to p-var-MC(FO), can be solved by $(f(k) \cdot \log n)$-space- and $f(k) \cdot n^{O(1)}$-time-bounded ATMs. Classical results from Chandra et al. [5] imply that FPT is the class of parameterized problems (P, κ) accepted by ATMs using space at most $f(k) + O(\log n)$. A similar fact holds for $(f(k) \cdot n^{O(1)})$-time-bounded DTMs that compute reductions; in this case we consider the problem of deciding whether a certain position in the output of the DTM contains a certain symbol. If the reduction is computed in time $f(k) \cdot n^{O(1)}$, then this problem can be decided by an ATM using space $f(k) + O(\log n)$. To finish the proof of the above claim, we (1) consider an ATM that model checks first-order formulas in space at most $f(k) \cdot \log n$ and time at most $f(k) \cdot n^{O(1)}$ with respect to the number of distinct variables as the parameter, and (2) modify it to run the above machine for the reduction whenever it wants to access an input symbol.

For the other direction, let (P, κ) be solvable by an ATM M using time $f(\kappa) \cdot n^c$ and space $f(\kappa) \cdot \log n$ for a computable function $f: \mathbb{N} \to \mathbb{N}$ and constant c on length-n inputs. In order to describe the reduction's construction we consider an input $x \in \Sigma^*$. We assume, without loss of generality, that M alternates between existential and universal states in each transition. That means, if we consider the start configuration as having time stamp 0, configurations with an even time stamp are always existential and configurations with an odd time stamp are always universal. Recall that a *configuration* C of M on input x consists of the current state, the head positions on the input tape and on the working tape, and the content of the working tape. Commonly a configuration is encoded as a (binary) string of length at most $c_M + f(k) \cdot \log n$ where c_M is a constant depending on M.

We present an fpt-reduction from (P, κ) to p-var-MC(FO). A first attempt for the reduction is to construct the (acyclic) *configuration graph* $G_M(x) = (\text{VERT}, \text{EDG})$ that contains all possible configurations of M as vertices and (directed) edges representing transitions between them. Moreover, the initial configuration is colored using a unary predicate I, and the existential and universal configurations are colored using unary predicates EXIST and UNIV, respectively. Then we state a formula φ_M that defines the acceptance behavior of ATMs, which is an alternating reachability query, that run in time at most $f(\kappa) \cdot n^c$ based on the graph. While we only need a constant number of variables for the formula, the graph considered in this reduction is too large to be constructible using an fpt-reduction since we consider all possible $2^{c_M + f(\kappa) \cdot \log n}$ configurations.

To get an fpt-reduction, we trade number of variables of the formula for the size of the constructed structure: Instead of constructing the graph explicitly, we modify the formula φ_M to a formula $\varphi_{M'}$ that not only defines the acceptance behavior of M, but also implicitly the configuration graph. How to modify the formula as well as how to construct a structure for this approach is described below.

Instead of constructing a configuration graph, the second version of our reduction produces a logical structure $\mathcal{A}$ with universe $U := \{1, \ldots, f(\kappa) \cdot n^c\}$. The only relation on these elements is the bit predicate $\textsc{bit} = \{(i, j) \mid \text{position } i \text{ in bit-string } \operatorname{enc}(j) \text{ is } 1\}$.

The formula φ_M uses variables to store pointers to whole configurations. In order to avoid this, we encode configurations of M using substrings for a configuration's state, head position, and working tape content. To encode a configuration with the help of a formula's element variables, we replace each element variable x in φ_M by a group of variables consisting of a single variable x_{state} to contain the index of the state (assuming that the input size is large enough), a variable $x_{\text{in-head}}$ to encode the head position on the input tape, a variable $x_{\text{work-head}}$ to encode the head position on the work tape, and $f(\kappa)$ variables $x_{\text{content-}i}$ to encode the content of the ith length-$\log n$ block on the working tape. Finally, we replace the predicate symbols that are used to access the edges and vertex colorings of the graph by subformulas that define these predicates based on the predicate $\textsc{bit}$. The details of this well-known approach from descriptive complexity are described in the book of Immerman [13].

Both the formula φ'_M and the used structure can be constructed in time $f(k) \cdot n^{O(1)}$, and the number of variables used in φ'_M is bounded in terms of the machine M and the parameter of the input instance k. ◀

Based on translating the alternating Turing machines from Lemma 5 into circuits, it is possible to get a characterization of SXP in terms of families of Boolean circuits that are uniform (the building blocks of the circuits can be recognized, for example, by using a parameterized version of the classical class ALOGTIME where the parameter is given in an appropriate way to the uniformity machine). These circuits have size $n^{f(k)}$ and depth $f(k) \cdot n^{O(1)}$. Thus, their shape also supports our intuition that SXP is the right analog of PSPACE in the parameterized setting.

5 Parameterized polynomial-space-complete problems

Kasai, Adachi and Iwata [14] introduced a simple pebble game to provide a combinatorial characterization of different complexity classes by playing several variants of that game. An instance of the pebble game consists of a set of nodes X, a set of start positions $S \subseteq X$ for $k = |S|$ pebbles, a goal node $\gamma \in X$ and a set $R \subseteq X^3$ of rules which are triples of pairwise distinct nodes. There are two players in the game, which alternately move a pebble on the game board according to some rule $(u, v, w) \in R$: if there are pebbles on u and v but not on w, then the corresponding player can move the pebble from u to w. One player wins the game if he puts a pebble on the goal node or reaches a position where the other player is unable to move. The game board is *acyclic* if the underlying dag with vertex set X and arcs (u, w), (v, w) for all $(u, v, w) \in R$ is acyclic. In the acyclic pebble game the game board is required to be acyclic. It turns out that determining the winner in the pebble game is complete for EXPTIME and determining the winner in the acyclic variant is PSPACE-complete [14]. If one fixes the number of pebbles k, it is possible to determine the winner (in both variants) in time $n^{O(k)}$. This can easily be verified as the game can be simulated by an alternating Turing machine that uses $O(k \log n)$ space to store the current position of the pebbles. Adachi, Iwata and Kasai [2] proved a corresponding lower bound in the non-acyclic case. They simulated single-tape Turing machines of running time $O(n^k)$ within the $(2k + 1)$-pebble game and used the time hierarchy theorem to obtain a lower bound of $n^{\Omega(k)}$. As remarked by Downey and Fellows [8] it follows that, parameterized by the number of pebbles, this problem is XP-complete. Thus, the pebble game supports the intuition that natural parameterizations of

EXPTIME-complete problems tend to be XP-complete. We show that the PSPACE-complete acyclic variant is complete for SXP under the same parameterization.

▶ **Theorem 6.** *Playing the pebble game on acyclic boards parameterized by the number of pebbles is complete for* SXP *under* fpt*-reductions.*

Proof. As the acyclic k-pebble game ends after a linear number of rounds, it can be simulated by an alternating Turing machine in space $O(k \log n)$ and time $O(n)$. Hence, this problem is in SXP. For the other direction we reduce from p-var-MC(FO). Let φ be a k-variable first-order formula and $\mathcal{A}$ be a structure with universe $[n]$. First, by allowing negation everywhere in the formula, we eliminate conjunction and universal quantification. We reduce the model checking problem to the acyclic $(k+2)$-pebble game such that $\mathcal{A} \models \varphi$ if, and only if, Player 1 wins the pebble game. We introduce a special node $\_$ to be used as middle vertex in the rules $(u, \_, w)$. At the beginning of the game there is a pebble on this node, which cannot be moved during the game. The acyclic game board resembles the structure of the formula. We use k pebbles to store the current assignment of the k variables and an additional pebble to control which subformula is evaluated. For every subformula ψ of φ we introduce a control node $X[\psi]$ and in addition nodes $X[\psi, x, v]$, for all variables $x \in \mathrm{var}(\varphi)$ and elements $v \in [n]$, to store assignments of the variables. These nodes serve as the basic data structure on the game board. We define the rules and additional nodes by induction on the structure of the formula to satisfy the following invariant.

For every subformula $\psi(x_1, \dots, x_k)$, Player 1 wins from the pebble position $(X[\psi]$, $X[\psi, x_1, v_1]$, ..., $X[\psi, x_k, v_k])$ iff $\mathcal{A} \models \psi[v_1, \dots, v_k]$.

As the pebble game is symmetric with respect to the players it follows that if the current pebble position is $(X[\psi]$, $X[\psi, x_1, v_1]$, ..., $X[\psi, x_k, v_k])$ and it is Player 2's turn, then Player 1 wins iff $\mathcal{A} \not\models \psi[v_1, \dots, v_k]$. Initially, the k value pebbles are on the nodes $X[\varphi, x_1, v]$, ..., $X[\varphi, x_k, v]$ for an arbitrary element v and the control pebble is on $X[\varphi]$. Thus, by the invariant above, Player 1 wins iff $\mathcal{A} \models \varphi$.

Atoms: For the base case let $\psi = R(x_{i_1}, \dots, x_{i_r})$ be an atom. We introduce additional nodes $Y[\psi, a_1, \dots, a_r]$ and $Y_p[\psi, x_{i_j}, a_1, \dots, a_r]$ for every tuple $(a_1, \dots, a_r) \in R^{\mathcal{A}}$, every variable x_{i_j}, $j \in [r]$ and $p \in \{1, 2\}$. There are rules $(X[\psi], \_, Y[\psi, a_1, \dots, a_r])$ that enable Player 1 to choose a tuple $(a_1, \dots, a_r)$ from the relation $R^{\mathcal{A}}$ that is consistent with the assignment specified by the pebbles on the nodes $X[\psi, x_i, v_i]$. To check this consistency both players are forced to use the following set of rules in the predefined order. First Player 2 moves the pebble from $Y[\psi, a_1, \dots, a_r]$ to $Y_1[\psi, x_{i_1}, a_1, \dots, a_r]$ using $(Y[\psi, a_1, \dots, a_r], \_, Y_1[\psi, x_{i_1}, a_1, \dots, a_r])$. Then it is Player 1's turn and both players alternately move the pebble using the rules $(Y_1[\psi, x_{i_j}, a_1, \dots, a_r]$, $X[\psi, x_{i_j}, a_j]$, $Y_2[\psi, x_{i_{j+1}}$, $a_1, \dots, a_r])$ and $(Y_2[\psi, x_{i_{j+1}}, a_1, \dots, a_r], \_, Y_1[\psi, x_{i_{j+1}}, a_1, \dots, a_r])$ for $j = 1, \dots, r-1$. Finally, there is a rule $(Y[\psi, x_{i_r}, a_1, \dots, a_r], \_, \gamma)$ that allows Player 1 to pebble the goal. By definition, this sequence of rules can be applied (and thus Player 1 wins the game as the goal node γ is pebbled) if $\mathcal{A} \models \psi[v_1, \dots, v_k]$. On the other hand, if $\mathcal{A} \not\models \psi[v_1, \dots, v_k]$, then Player 1 gets stuck and Player 2 wins.

Disjunction: If $\psi = \psi_1 \vee \psi_2$, then we have to ensure that from the pebble position $(X[\psi], X[\psi, x_1, v_1], \dots, X[\psi, x_k, v_k])$ Player 1 can move to either $(X[\psi_1], X[\psi_1, x_1, v_1], \dots,$ $X[\psi_1, x_k, v_k])$ or $(X[\psi_2], X[\psi_2, x_1, v_1], \dots, X[\psi_2, x_k, v_k])$. To make this decision, we introduce nodes $X_p[\psi_j]$, for $p, j \in \{1, 2\}$, and rules $(X[\psi], \_, X_1[\psi_j])$ and $(X_1[\psi_j], \_, X_2[\psi_j])$ for $j \in \{1, 2\}$. Thus, Player 1 can choose an j and move to $X_1[\psi_j]$. Afterwards, Player 2 is forced to move to $X_2[\psi_j]$. It remains to copy the current assignment to the subformula, that is, the players have to move the pebbles from $X[\psi, x, v]$ to $X[\psi_j, x, v]$. We use nodes $X[\psi_j, i]$ for $i \in [k+1]$ to control this process. The first rule is $(X_2[\psi_j], \_, X[\psi_j, 1])$. For all $v \in [n]$ and $i \leq k$ there are rules $(X[\psi, x_i, v], X[\psi_j, i], X[\psi_j, x_i, v])$ for copying the value of x_i and $(X[\psi_j, i], X[\psi_j, v, x_i], X[\psi_j, i+1])$ to move the control pebble. Both players must cooperate and use these rules successively to move the pebbles from $X[\psi, x, v]$ to $X[\psi_j, x, v]$. At the end of this process the pebble position is $(X[\psi_j, k+1], X[\psi_j, x_1, v_1]), \dots, X[\psi_j, x_k, v_k])$ and it is Player 2's turn. He is forced to use the final rule $(X[\psi_j, k+1], \_, X[\psi_j])$. This finishes the description of the $\vee$-case.

Negation: Let $\psi = \neg\psi'$. We simulate negation by changing the role of both players. That is, from the configuration $(X[\psi], X[\psi, x_1, v_1], \dots, X[\psi, x_k, v_k])$ where it is Player 1's turn we want to force both players to reach the configuration $(X[\psi'], X[\psi', x_1, v_1], \dots, X[\psi', x_k, v_k])$ where it is Player 2's turn. Changing is role of the players is rather easy as we just have to introduce a dummy rule $(X[\psi], \_, X[\psi'])$ to force one move of the control pebble. Afterwards the players have to move the assignment pebbles from $X[\psi, x, v]$ to $X[\psi', x, v]$. This copy process can be done in the same deterministic way as described in the $\vee$-case.

Existential quantification: Let $\psi = \exists x_j \psi'$. To model the existential quantifier we have to ensure that from a pebble position $(X[\psi], X[\psi, x_1, v_1], \dots, X[\psi, x_j, v_j]), \dots, X[\psi, x_k, v_k])$ Player 1 can choose an element $w \in [n]$ and reach the new position $(X[\psi'],\ X[\psi', x_1, v_1],$ $\dots,\ X[\psi', x_j, w]),\ \dots,\ X[\psi, x_k, v_k])$. Copying the values of x_i for $i \neq j$ (moving the pebbles from $X[\psi, x_i, v_i]$ to $X[\psi', x_i, v_i]$) can be done in the same way as in the previous cases. Hence, we end up with a configuration where it is Player 1's turn and the pebbles are on $X[\psi', x_i, v_i]$ $(i \neq j)$, $X[\psi, x_j, v_j]$, and the control pebble is on an additional node $X_0[\psi]$. To change the value of x_j we use the following construction. Let $X_1[\psi]$ and $X_2[\psi]$ be two additional nodes. There is a rule $(X_0[\psi], \_, X_1[\psi])$ and for every $v \in [n]$ we add the following three rules: $(X[\psi, x_j, v], X_1[\psi], X_2[\psi])$, $(X_1[\psi], X_2[\psi], X[\psi', x_j, v])$, $(X_2[\psi], X[\psi', x_j, v], X[\psi'])$. First, Player 1 moves the pebble from $X_0[\psi]$ to $X_1[\psi]$. Afterwards, Player 2 is forced to move the pebble from $X[\psi, x_j, v_j]$ to $X_2[\psi]$. Now Player 1 can choose some $w \in [n]$ and move the pebble from $X_1[\psi]$ to $X[\psi', x_j, w]$. The last move is done by Player 2, who is forced to move the pebble from $X_2[\psi]$ to $X[\psi']$. ◀

Another example that shifts the correspondence between EXPTIME and PSPACE in the classical world to XP and SXP in the parameterized setting is resolution. Resolution is a well-known and intensively studied proof system to detect the unsatisfiability of a given formula in conjunctive normal form. Starting with the clauses from the CNF formula one iteratively derives new clauses using only one simple rule: The resolution rule for a variable X takes two clauses $\gamma \cup \{X\}$, $\delta \cup \{\neg X\}$ and resolves $\gamma \cup \delta$. The given CNF formula is unsatisfiable if, and only if, the empty clause can be derived. The *width* of a refutation is the maximal number of literals in every clause of the derivation. A resolution derivation can naturally be viewed as a directed acyclic graph (dag) where the nodes are labeled with clauses and arcs pointing from the resolvent to its parents. The *depth* of a refutation is the length of the longest path in the corresponding dag. If on every path in this derivation dag no variable has been used twice by the resolution rule, then the derivation is *regular*. Note that the depth of every regular resolution refutation is at most linear in the number of variables, thus *linear depth resolution* generalizes regular resolution.

A resolution refutation of width k can be found be an alternating $O(k \log n)$-space Turing machine as follows: In each step, the machine stores one clause (using $k \log n$ bits) and tries to justify that this clause can be derived. Starting from the empty clause, the machine existentially guesses its parents and then universally chooses a parent to justify that it can be derived. The machine accepts the input, if the current clause is from the CNF formula

Table 1 To compare the correspondence between tractable and intractable in classical and parameterized complexity we denote by *poly* and *exp* polynomial and exponential growth, and by *fpt* and *xp* growth of the form $f(k) \cdot n^{O(1)}$ and $n^{f(k)}$, respectively.

Alternating Turing machines	EXPTIME	Alternating PSPACE machines of *exp* running time.
	PSPACE	Alternating PSPACE machines of *poly* running time.
	XP	Alternating SPACE($f(k)\log n$) machines of *xp* running time.
	SXP	Alternating SPACE($f(k)\log n$) machines of *fpt* running time.
Pebble games	EXPTIME	Pebble game.
	PSPACE	Acyclic pebble game.
	XP	Pebble game, parameter: the number of pebbles.
	SXP	Acyclic pebble game, parameter: the number of pebbles.
Resolution	EXPTIME	Bounded width resolution.
	PSPACE	Bounded width linear depth resolution.
	XP	Bounded width resolution, parameter: the width.
	SXP	Bounded width linear depth resolution, parameter: the width.

and rejects after $(2n)^k$ steps (which upper bounds the depth of width-k refutations). If we additionally require the depth to be linear, the alternating machine is able to find a refutation of width k in linear time. It follows that finding resolution refutations of width k is in EXPTIME, if k is part of the input, and in XP, if k is the parameter. Furthermore, finding linear depth refutations of width k is in PSPACE, if k is part of the input, and in SXP, if k is the parameter. By reducing the pebble game to bounded width resolution [3] it was shown that the corresponding problems are complete for EXPTIME, XP and PSPACE. We now show that the same reduction, stated in Fact 7, can be used to show that finding linear depth resolution refutations of bounded width is SXP-complete.

▶ **Fact 7** ([3]). *There is an* fpt*-reduction that takes an instance of the k-pebble game and produces a 3-*CNF *formula Γ such that Player 1 wins the k-pebble game iff Γ has a resolution refutation of width $k+1$. If in addition the game board is acyclic, then Γ has a regular resolution refutation of width $k+1$.*

▶ **Lemma 8.** *Finding linear depth resolution refutations of width k is complete for* SXP*.*

Proof. We already have observed that this problem is contained in SXP. Note that the reduction stated in Fact 7 reduces the parametrized acyclic pebble game to parameterized linear depth resolution, as every regular refutation has always linear depth. By Theorem 6 it follows that finding linear depth refutations is complete for SXP when parameterized by the width. ◀

6 Conclusion

We placed the model checking problem for fixed variable first-order logic within the hierarchy of intractable problems in parameterized complexity. As a consequence we exhibited a new parameterized complexity class, SXP, that corresponds to PSPACE in the same way as XP corresponds to EXPTIME. To support this intuition we gave characterizations in terms of alternating Turing machines, pebble games, and resolution refutations. The results are summarized in Table 1.

References

1 Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73(3):235–276, 1995.

2 Akeo Adachi, Shigeki Iwata, and Takumi Kasai. Some combinatorial game problems require $\Omega(n^k)$ time. *J. ACM*, 31:361–376, March 1984.

3 Christoph Berkholz. On the complexity of finding narrow proofs. In *Foundations of Computer Science, IEEE Annual Symposium on*, pages 351–360, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

4 Christoph Berkholz. Lower bounds for existential pebble games and k-consistency tests. *Logical Methods in Computer Science*, 9(4), 2013.

5 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

6 Yijia Chen, Jörg Flum, and Martin Grohe. Machine-based methods in parameterized complexity theory. *Theoretical Computer Science*, 339(2–3):167–199, 2005.

7 R. Downey and M. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

9 Rodney G. Downey, Michael R. Fellows, and Ken Regan. Descriptive complexity and the W-hierachy. In *Proof Complexity and Feasible Arithmetic*, volume 39 of *AMS-DIMACS*, pages 119–134. AMS, 1998.

10 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comp.*, 31(1):113–145, 2001.

11 Jörg Flum and Martin Grohe. Model-checking problems as a basis for parameterized intractability. *Logical Methods in Computer Science*, 1(1), 2005.

12 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

13 Neil Immerman. *Descriptive complexity*. Springer, New York, 1999.

14 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM J. Comput.*, 8(4):574–586, 1979.

15 Richard E Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7:18–20, 1975.

16 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.

17 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

18 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1982)*, pages 137–146. ACM, 1982.

Synchronizing Words for Weighted and Timed Automata*

Laurent Doyen[1], Line Juhl[2], Kim G. Larsen[2], Nicolas Markey[1], and Mahsa Shirmohammadi[1,3]

1 **Laboratoire Spécification & Vérification – CNRS & ENS Cachan, France**
2 **CISS – Aalborg University, Denmark**
3 **Dpt Informatique – Université Libre de Bruxelles, Belgium**

Abstract

The problem of synchronizing automata is concerned with the existence of a word that sends all states of the automaton to one and the same state. This problem has classically been studied for complete deterministic finite automata, with the existence problem being NLOGSPACE-complete.

In this paper we consider synchronizing-word problems for weighted and timed automata. We consider the synchronization problem in several variants and combinations of these, including deterministic and non-deterministic timed and weighted automata, synchronization to unique location with possibly different clock valuations or accumulated weights, as well as synchronization with a safety condition forbidding the automaton to visit states outside a safety-set during synchronization (e.g. energy constraints). For deterministic weighted automata, the synchronization problem is proven PSPACE-complete under energy constraints, and in 3-EXPSPACE under general safety constraints. For timed automata the synchronization problems are shown to be PSPACE-complete in the deterministic case, and undecidable in the non-deterministic case.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases Synchronizing words, weighted automata, timed automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.121

1 Introduction

The notion of synchronizing automata is concerned with the following natural problem: *how can we regain control over a device if we do not know its current state?* Since losing the control over a device may happen due to missing the observation on the outputs produced by the system, static strategies, which are finite sequences (or words) of input letters are considered while synchronizing systems. As an example think of remote systems connected to a wireless controller that emits the command via wireless waves but expects the observations via physical connectors (it might be excessively expensive to mount wireless senders on the remote systems), and consider that the physical connection to the controller is lost because of some technical failure. The wireless controller can therefore not observe the current states of distributed subsystems. In this setting, emitting a synchronizing word as the command leaves the remote system (as a whole) in one particular state, no matter which state each distributed subsystem started at; thus the controller can regain control. For synchronizing automata, there are also applications e.g. in planning, control of discrete event systems, bio-computing, and robotics [2, 8, 4].

* This work was partially supported by the Belgian Fonds National de la Recherche Scientifique (FNRS), by FP7 projects Cassting (601148) and ERC EQualIS (308087).

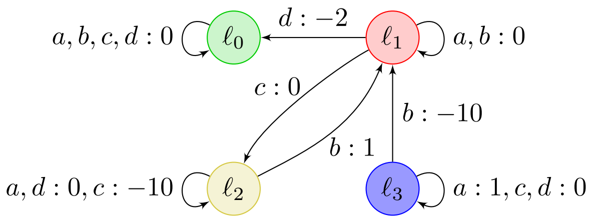


Figure 1 A complete deterministic WA with location-synchronizing word $a^{10} \cdot b \cdot (c \cdot b)^2 \cdot d$ under non-negative safety condition.

Synchronizing automata have classically been studied in the setting of complete deterministic finite-state automata, with polynomial bounds on the length of the shortest synchronizing word [3] and the existence problem being NLOGSPACE-complete. In this paper, we consider synchronization in systems whose behavior depends on quantitative constraints. We study two classes of such systems, weighted automata (WAs) and timed automata (TAs), and introduce variants of synchronization to include the quantitative aspects as well as some safety condition while synchronizing. The main challenge is that we are now facing automata with infinite state-spaces and infinite branching (e.g. delays in a TA).

For WAs, states are composed of locations and quantitative weights. As weights are merely accumulated in this setting, it is impossible to synchronize to a single state. Instead we search for a *location-synchronizing word*, i.e., a word after which all states will agree on the location. In addition, we add a safety condition insisting that during synchronization the accumulated weight (energy) is *safe*, e.g. a non-negative safety condition that requires the system to never run out of power while synchronizing. Considering the safety condition is what distinguishes our setting from the one presented in [6]; moreover, in that work WAs are restricted to have only non-negative weights on transitions. Figure 1 illustrates a WA with four locations and four letters. We have to synchronize infinitely many states (ℓ_i, e) where ℓ_i is one of the four locations and $e \in \mathbb{R}$ is the accumulated energy. The only way to location-synchronize a state (ℓ_3, e) with states involving other locations is to input b. However, if b is provided initially, this will drop the energy level by -10 violating the non-negative safety condition for $(\ell_3, 0)$. Fortunately, the letter a recharges the energy level at ℓ_3 and has no negative effect at other locations. After reading $a^{10}b$, all states are synchronized in ℓ_0 and ℓ_1 with energy at least 0. Next, a d-input can location-synchronize states involving ℓ_0 and ℓ_1, but it drops the energy level at ℓ_1 by -2. Again, we try to find a word that recharges the energy at ℓ_1. Supplying $c \cdot b$ twice makes a d-transition safe to be taken to location-synchronize safe states involving ℓ_0 and ℓ_1. So, the word $a^{10} \cdot b \cdot (c \cdot b)^2 \cdot d$ location-synchronizes the automaton with non-negative safety condition.

For TAs, synchronizing the classical region abstraction is not sound. Figure 2 displays a 1-letter TA with four locations. We have infinitely many states to synchronize using the letter a and quantitative delays $\mathsf{d}(t)$ ($t \in \mathbb{R}_{\geq 0}$). We propose an algorithm which first reduces the (uncountably) infinite set of configurations into a finite set (with at most the number of locations in the TA), and then pairwise synchronizes the obtained finite set of states. The word $\mathsf{d}(3) \cdot a \cdot a$ is a *finitely synchronizing word* that synchronizes the infinite set of states into a finite set: whatever the initial state, inputting the word $\mathsf{d}(3) \cdot a \cdot a$ the TA ends up in one of the states $(\ell_0, 0)$, $(\ell_1, 0)$ or $(\ell_3, 0)$. Moreover, since ℓ_3 cannot be escaped, any synchronizing word in this automaton lead to a state involving ℓ_3. It then suffices to play $a \cdot \mathsf{d}(1) \cdot a \cdot a \cdot a$ to end up in $(\ell_3, 0)$, whatever the initial state. A possible synchronizing word for this TA is then $\mathsf{d}(3) \cdot a^3 \cdot \mathsf{d}(1) \cdot a^3$, which always leads to the state $(\ell_3, 0)$.

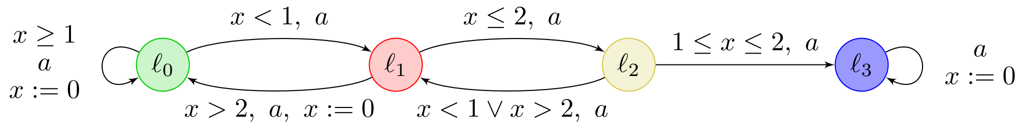


Figure 2 A complete deterministic 1-letter TA with synchronizing word $\mathsf{d}(3) \cdot a^3 \cdot \mathsf{d}(1) \cdot a^3$.

In this paper we consider the synchronization problem for TAs and WAs in several variants: including deterministic and non-deterministic TAs and WAs, synchronization to unique location with possibly different clock valuations or accumulated weights, as well as synchronization with a safety condition forbidding the automaton to visit states outside a safety-set during synchronization (e.g. energy constraints). Our results can be seen in Table 1. For TAs the synchronization problems are shown to be PSPACE-complete in the deterministic case, and undecidable in the non-deterministic case. For deterministic WAs, the synchronization problem is proven PSPACE-complete under energy constraints, and in 3-EXPSPACE under general safety constraints.

The detailed proofs of these results can be found in a full version of this paper [5].

2 Definitions

A *labeled transition system* over a (possibly infinite) alphabet Γ is a pair $\langle Q, R\rangle$ where Q is a set of states and $R \subseteq Q \times \Gamma \times Q$ is a transition relation. The labeled transition systems we consider have state space $Q = L \times X$ consisting of a finite set L of locations and a possibly infinite set X of quantitative values. Given a state $q = (\ell, x)$, let $\mathsf{loc}(q) = \ell$ be the location of q, and for $a \in \Gamma$, let $\mathsf{post}(q, a) = \{q' \mid (q, a, q') \in R\}$. For $P \subseteq Q$, let $\mathsf{loc}(P) = \{\mathsf{loc}(q) \mid q \in P\}$ and $\mathsf{post}(P, a) = \bigcup_{q \in P} \mathsf{post}(q, a)$. For nonempty words $w \in \Gamma^+$, define inductively $\mathsf{post}(q, aw) = \mathsf{post}(\mathsf{post}(q, a), w)$. A *run* (or *path*) in a labeled transition system $\langle Q, R\rangle$ over Γ is a finite sequence $q_0 q_1 \cdots q_n$ such that there exists a word $a_0 a_1 \cdots a_{n-1} \in \Gamma^*$ for which $(q_i, a_i, q_{i+1}) \in R$ for all $0 \leq i < n$.

Synchronizing words

A word $w \in \Gamma^+$ is *synchronizing* in the labeled transition system $\langle Q, R\rangle$ if $\mathsf{post}(Q, w)$ is a singleton, and it is *location-synchronizing* if $\mathsf{loc}(\mathsf{post}(Q, w))$ is a singleton. Given $U \subseteq Q$, a word w is synchronizing (resp., location-synchronizing) in $\langle Q, R\rangle$ *with safety condition* U if $\mathsf{post}(U, w)$ is a singleton (resp., $\mathsf{loc}(\mathsf{post}(U, w))$ is a singleton) and $\mathsf{post}(U, v) \subseteq U$ for all prefixes v of w. Thus a synchronizing word can be read from every state and bring the system to a single state, and a location-synchronizing word brings the system to a single location, possibly with different quantitative values. The safety condition U requires that the states in $Q \setminus U$ are never visited while reading the word. In this paper, we specify the safety condition U by a function $\mathsf{Safe}\colon L \to X$, then $U = \{(\ell, x) \in Q \mid x \in \mathsf{Safe}(\ell)\}$. We say that a system is (location-)synchronizing if it has a (location-)synchronizing word. The (location-)synchronizing problem (under a safety condition) asks, given a system (and a safety condition), whether the system is (location-)synchronizing.

A finite state automaton is a special kind of labeled transition systems where the alphabet and the state space are finite. Synchronizing words of finite-state automata have already been extensively studied. The synchronizing problem in a finite-state automaton $\mathcal{A}$ is easily reduced to a reachability problem in the power-set automaton of $\mathcal{A}$. This provides a PSPACE algorithm for this problem, and the problem is proved PSPACE-complete [7]. When $\mathcal{A}$ is

Table 1 Summary of obtained results.

			Timed Automata (TAs)	Weighted Automata (WAs)
Deterministic	No condition	Synchronization	PSPACE-complete	Trivial (always `false`)
		Loc.-synchronization	PSPACE-complete	NLOGSPACE-complete
	Safety condition	Synchronization	?	PSPACE-complete
		Loc.-synchronization	?	3-EXPSPACE energy cond.: PSPACE-c.
Non-deterministic	No condition	Synchronization	Undecidable	Trivial (always `false`)
		Loc.-synchronization	Undecidable	PSPACE-complete
	Safety condition	Synchronization	Undecidable	PSPACE-complete
		Loc.-synchronization	Undecidable	?

deterministic and complete, that means $|\mathsf{post}(q,a)| = 1$ for all states q and letters a, a better algorithm is obtained by iteratively synchronizing pairs of states [3, 8]: the existence of a synchronizing word in $\mathcal{A}$ is indeed equivalent to the existence of synchronizing words for each pair of states of $\mathcal{A}$, which is reduced to polynomially-many reachability problems in the product of two copies of $\mathcal{A}$. The problem can then be proven NLOGSPACE-complete.

We consider labeled transition systems induced by WAs and TAs. We are interested in (location-)synchronizing problem (with or without safety condition) in the labeled transition systems induced by TAs and WAs, defined below.

Weighted automata (WAs)

A *weighted automaton* (WA) over a finite alphabet Σ is a tuple $\mathcal{A} = \langle L, E\rangle$ consisting of a finite set L of locations, and a set $E \subseteq L \times \Sigma \times \mathbb{Z} \times L$ of edges. When E is clear from the context, we denote by $\ell \xrightarrow{a:z} \ell'$ the edge $(\ell, a, z, \ell') \in E$, which represents a transition on letter a from location ℓ to ℓ' with weight z. We view the weights as the resource (or energy) consumption of the system. The semantics of a WA $\mathcal{A} = \langle L, E\rangle$ is the labeled transition system $[\![\mathcal{A}]\!] = \langle Q, R\rangle$ on the alphabet $\Gamma = \Sigma$ where $Q \subseteq L \times \mathbb{Z}$ and $((\ell, e), a, (\ell', e')) \in R$ if $(\ell, a, e' - e, \ell') \in E$. In a state (ℓ, e), we call e the *energy level.* The WA $\mathcal{A}$ is *deterministic* if for all edges (ℓ, a, z_1, ℓ_1), $(\ell, b, z_2, \ell_2) \in E$, if $a = b$, then $z_1 = z_2$ and $\ell_1 = \ell_2$; it is *complete* if for all $\ell \in L$ and all $a \in \Sigma$, there exists an edge $(\ell, a, z, \ell') \in E$.

Let $\mathcal{I}$ be the set of intervals with integer or infinite endpoints. For WAs, we consider safety conditions of the form $\mathsf{Safe}\colon L \to \mathcal{I}$, and we denote an interval $[y, z]$ by $y \leq e \leq z$, an interval $[z, +\infty)$ by $e \geq z$, etc. where e is an energy variable.

Timed automata (TAs)

Let $C = \{x_1, \ldots, x_{|C|}\}$ be a finite set of *clocks.* A (clock) valuation is a mapping $v\colon C \to \mathbb{R}_{\geq 0}$ that assigns to each clock a non-negative real number. We denote by $\mathbf{0}_C$ (or $\mathbf{0}$ when the set of clocks is clear from the context) the valuation that assigns 0 to every clock.

A *guard* $g = (I_1, \ldots, I_{|C|})$ over C is a tuple of $|C|$ intervals $I_i \in \mathcal{I}$. A valuation v satisfies g, denoted $v \models g$, if $v(x_i) \in I_i$ for all $1 \leq i \leq |C|$. For $t \in \mathbb{R}_{\geq 0}$, we denote by $v + t$ the valuation defined by $(v + t)(x) = v(x) + t$ for all $x \in C$, and for a set $r \subseteq C$ of clocks, we denote by $v[r]$ the valuation such that $v[r](x) = 0$ for all $x \in r$, and $v[r](x) = v(x)$ otherwise.

A *timed automaton* (TA) over a finite alphabet Σ is a tuple $\langle L, C, E\rangle$ consisting of a finite set L of locations, a finite set C of clocks, and a set $E \subseteq L \times \mathcal{I}^{|C|} \times \Sigma \times 2^C \times L$ of edges. When E is clear from the context, we denote by $\ell \xrightarrow{g,a,r} \ell'$ the edge $(\ell, g, a, r, \ell') \in E$, which represents a transition on letter a from location ℓ to ℓ' with guard g and set r of clocks to reset. The semantics of a TA $\mathcal{A} = \langle L, C, E\rangle$ is the labeled transition system $[\![\mathcal{A}]\!] = \langle Q, R\rangle$ over the alphabet $\Gamma = \mathbb{R}_{\geq 0} \cup \Sigma$ (assuming $\Sigma \cap \mathbb{R}_{\geq 0} = \emptyset$) where $Q = L \times (C \to \mathbb{R}_{\geq 0})$, and $((\ell, v), \gamma, (\ell', v')) \in R$ if

- either $\gamma \in \mathbb{R}_{\geq 0}$, and $\ell = \ell'$ and $v' = v + \gamma$;
- or $\gamma \in \Sigma$, and there is an edge $(\ell, g, \gamma, r, \ell') \in E$ such that $v \models g$ and $v' = v[r]$.

The TA $\mathcal{A}$ is *deterministic* if for all states $(\ell, v) \in Q$, for all edges $(\ell, g_1, a, r_1, \ell_1)$ and $(\ell, g_2, b, r_2, \ell_2)$ in E, if $a = b$, and $v \models g_1$ and $v \models g_2$, then $r_1 = r_2$ and $\ell_1 = \ell_2$; it is *complete* if for all $(\ell, v) \in Q$ and all $a \in \Sigma$, there exists an edge $(\ell, g, a, r, \ell') \in E$ such that $v \models g$.

3 Synchronization in deterministic WAs

In this section, we prove that location-synchronizing problem for deterministic WAs is decidable. In the absence of safety conditions, two states involving the same location but different initial energy can never be synchronized (synchronizing problem is trivial); however in that setting, location-synchronization is equivalent to synchronization of deterministic finite-state automata (i.e. weights play no role). In the presence of safety conditions, synchronization is also most-often impossible, for the same reason as above. The only exception is when safety condition is punctual (at most one safe energy level for each location), in which case the problem becomes equivalent to synchronizing partial (not-complete) finite-state automata, which is PSPACE-complete [7]. We thus focus on location-synchronization with safety conditions. We fix a complete deterministic WA $\mathcal{A} = \langle L, E\rangle$ over the alphabet Σ, where the maximum absolute value appearing as weight in transitions is Z.

3.1 Location-synchronization under lower-bounded safety condition

In this subsection we assume that all the locations have safety conditions of the form $e \geq n$, with $n \in \mathbb{Z}$. This is equivalent to having only safety conditions of the form $e \geq 0$: it suffices to add $-n$ to the weight of all incoming transitions and to add $+n$ to the weight of outgoing transitions. In the sequel, we consider safety conditions of the form $e \geq 0$, which we call *non-negative safety conditions* or *energy condition*.

▶ **Theorem 1.** *The existence of a location-synchronizing word in $\mathcal{A}$ under non-negative safety condition* Safe *is* PSPACE*-complete.*

Proof. Runs starting from two states with same location but two different energy levels $e_2 > e_1$, always go through the states involving the same locations and the energy levels preserving the difference $e_2 - e_1$. Therefore, to decide whether $\mathcal{A}$ is location-synchronizing under non-negative safety condition, it suffices to check if there is a word that synchronizes all locations with the initial energy $\mathbf{0}$, into a single location. We show that deciding whether such word w exists is in PSPACE by providing an upper bound for the length of w.

Below, we assume that $\mathcal{A}$ has a location-synchronizing word. For all subsets $S \subseteq L$ with cardinality $m > 2$, there is a word that synchronizes S into some strictly smaller set. To characterize the properties of such words, we consider the weighted digraph G_m induced by the product between m copies of $\mathcal{A}$, where all vertices in $\{(\ell, \dots, \ell) \mid \ell \in L\}$, which are vertices with m identical locations, are replaced with a new vertex synch. All ingoing transitions to some location in $\{(\ell, \dots, \ell) \mid \ell \in L\}$ are redirected to synch. There is only a self-loop transition in synch. An edge with weight $\langle z_1, \dots, z_m \rangle$ is *non-negative* (resp., *zero-effect*) if $z_i \geq 0$ for all dimensions $1 \leq i < m$ (resp., $z_i = 0$); and it is *negative* otherwise. A non-negative edge is *positive* if z_i is positive for some dimension i. There is a one-to-one correspondence between a path $x_0 x_1 \cdots x_n$ in G_m and a group of m runs $\rho^1 \dots \rho^m$ in $\mathcal{A}$ such that all runs ρ^i are in shape of $\rho^i = \ell_0^i \cdots \ell_n^i$ where $x_j = (\ell_j^1, \dots, \ell_j^m)$ for all $0 \leq j \leq n$. A path is *safe* if all corresponding m runs ρ^i starting from ℓ_0^i with energy level $\mathbf{0}$, always keep a non-negative energy level while going through all the locations $\ell_1^i \cdots \ell_n^i$ along the run.

The following lemma is a key to compute an upper bound for the length of location-synchronizing words. Roughly speaking, it states that for all subsets S of locations, either there is a *short* word w that synchronizes S into a strictly smaller set, or there exists a family of words $w_0 \cdot (w_1)^i$ $(i \in \mathbb{N})$ such that inputting the word $w_0 \cdot (w_1)^i$ accumulates energy i for the run starting in some location $\ell \in S$, while having non-negative effects along the runs starting from the other locations in S. Consider the WA depicted in Fig. 1. Since in the digraph G_2, there is no safe path from (ℓ_0, ℓ_2) to synch, there is a family of words $(b \cdot c)^i$ such that each iteration of $b \cdot c$ increase the energy level in ℓ_2 by 1.

▶ **Lemma 2.** *For all $1 < m \leq |L|$, for all vertices x of the digraph G_m, there is either a safe simple path from x to synch, or a simple cycle where all edges are non-negative and at least one is positive, which is reachable from x via a safe path.*

The next lemma states that $\mathcal{A}$ has a location-synchronizing word if it has a *short* one, of length at most $\mathsf{Z}^{|L|} \times |L|^{3+|L|^2}$. Since this value can be stored in polynomial space, an (N)PSPACE algorithm can decide whether the given WA is location-synchronizing.

▶ **Lemma 3.** *For the synchronizing WA $\mathcal{A}$, there exists a short location-synchronizing word.*

To show PSPACE-hardness, we use a reduction from synchronizing word problem for deterministic finite automata with partially defined transition function that is PSPACE-complete [7]. From a partial finite state automaton $\mathcal{A}$, we construct a WA $\mathcal{A}'$. All defined transitions of $\mathcal{A}$ are augmented with the weight 0 in $\mathcal{A}'$. To complete $\mathcal{A}'$, all non-defined transitions are added but with weight -1. Since the safety condition is non-negative in all locations, none of the transitions with weight -1 are allowed to be used along synchronization in $\mathcal{A}'$. So, $\mathcal{A}$ has a synchronizing word if, and only if, $\mathcal{A}'$ has a location-synchronizing one. ◀

We generalize the synchronizing word problem to *location-synchronization from a subset*, where the aim is to synchronize a given subset of locations. This variant is used to decide location-synchronization under general safety condition. Given a subset $S \subseteq L$ of locations, we prove Lemma 4 using reductions from and to coverability in vector-addition systems.

▶ **Lemma 4.** *Deciding the existence of a location-synchronizing word from S in $\mathcal{A}$ under lower-bounded safety condition* Safe *is decidable in* 2-EXPSPACE*, and it is* EXPSPACE*-hard.*

3.2 Location-synchronization under general safety condition

We now discuss location-synchronization under the *general* safety condition where the energy constraint for each location can be a bounded interval, lower or upper-bounded, or trivial (always `true`). We proceed in two steps: first, we prove that the PSPACE-completeness results in case of energy safety condition is preserved in location-synchronization under safety

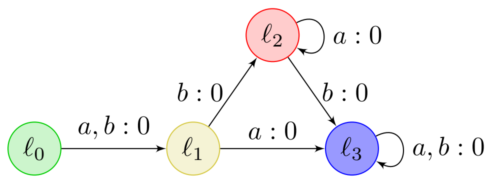


Figure 3 To location-synchronize the automaton, taking the back-edge $\ell_3 \xrightarrow{b,0} \ell_2$ is avoidable.

condition with only lower-bounded or trivial constraints. Second, we extend our techniques to establish results for general safety conditions. To obtain results for the general case, we use the variant of *location-synchronization from a subset*, that is discussed in all cases too.

Location-synchronization under lower-bounded or trivial safety conditions

Let the safety condition Safe assign to each location of L either an interval of the form $[n, +\infty)$ or `true`, and let us partition L into two classes $L_{\mapsto}$ and $L_{\leftrightarrow}$ accordingly. A *back-edge* is a transition that goes from a location in $L_{\leftrightarrow}$ to a location in $L_{\mapsto}$. Consider the WA drawn in Fig. 3 with four locations and two letters. The safety condition is non-negative in ℓ_0 and ℓ_2, and is trivial in ℓ_1 and ℓ_3: $L_{\mapsto} = \{\ell_0, \ell_2\}$ and $L_{\leftrightarrow} = \{\ell_1, \ell_3\}$. Thus, the transition $\ell_1 \xrightarrow{b:0} \ell_2$ is a back-edge. The word abb is a location-synchronizing word that takes the back-edge $\ell_1 \xrightarrow{b:0} \ell_2$ in ℓ_1 (with non-negative energy levels). In this example, there exists an alternative word aab that takes no back-edges and still location-synchronizes the automaton. We prove, by Lemma 5, that such words always exist implying that taking back-edge transitions while synchronizing is avoidable in deterministic WAs.

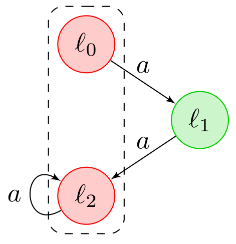
▶ **Lemma 5.** *There is a location-synchronizing word in $\mathcal{A}$ under lower-bounded or trivial safety condition* Safe *if, and only if, there is one in the automaton obtained from $\mathcal{A}$ by removing all back-edge transitions.*

Lemma 5 does not hold when synchronizing from a subset S of the locations. Indeed, consider the one-letter automaton of Fig. 4: the locations ℓ_0 and ℓ_2 have non-negative safety condition, while the location ℓ_1 has trivial safety condition. Obviously, it is possible to location-synchronize from the set $S = \{\ell_0, \ell_2\}$, and this would not be possible without taking the back-edge $\ell_1 \xrightarrow{a} \ell_2$. The result also fails for non-deterministic WAs. Consider the WA depicted in Fig. 5, where $L_{\mapsto} = \{1, 2\}$ and $L_{\leftrightarrow} = \{3, 4\}$. We claim that the back-edge $3 \xrightarrow{b:+1} 2$ is needed to location-synchronize. Initially, only letter a is available, because b corresponds to a back-edge from 3 to 2 and would violate the safety condition there, while the c-transition from 2 to 1 violates the condition in the location 1. After this step, inputting more a's is possible, but would not modify the set of states that have been reached, and in particular would not help synchronizing. inputting c is still not an option (the same reason as previously), so that only b is interesting, resulting in a back-edge. It remains to ensure that there is indeed a way of synchronizing into the location 4, which is inputting c twice.

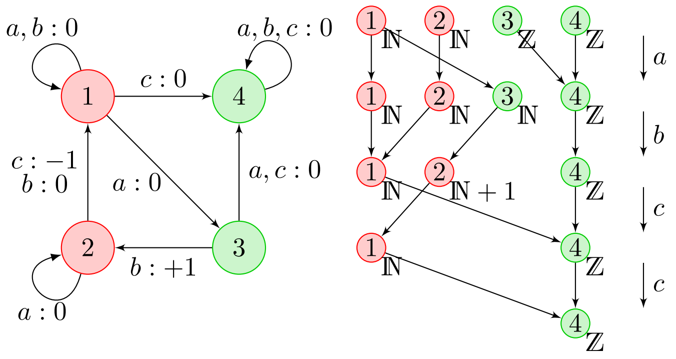
In the absence of back-edges and with non-empty $L_{\leftrightarrow}$, location-synchronization can be achieved in two steps: first location-synchronize all the states of $L_{\mapsto}$ to some location in $L_{\leftrightarrow}$ using Theorem 1; then location-synchronize the states in $L_{\leftrightarrow}$ where the weights play no role.

▶ **Lemma 6.** *The existence of a location-synchronizing word in $\mathcal{A}$ under lower-bounded or trivial safety condition* Safe *is* PSPACE*-complete.*

The proof of Lemma 6 carries on for synchronizing from a subset of locations, except using Lemma 4 instead of Theorem 1, and requiring that the automaton has no back-edge.



■ **Figure 4** Unavoidable back-edges to synchronize from a subset.



■ **Figure 5** Unavoidable back-edges to synchronize non-deterministic WA.

▶ **Lemma 7.** *Assume that $\mathcal{A}$ has no back-edge, and pick a set S of locations such that $L_{\leftrightarrow} \subseteq S$. The existence of a location-synchronizing word in $\mathcal{A}$ from S under lower-bounded or trivial safety condition* Safe *is decidable in* 2-EXPSPACE, *and it is* EXPSPACE-*hard.*

Location-synchronization under general safety conditions

Let us relax the constraints on the safety condition Safe: some locations may have bounded intervals to indicate the safe range of energy. The set L of locations is partitioned into $L_{\mapsto\!\!\!\dashv}$, $L_{\mapsto}$ and $L_{\leftrightarrow}$ where locations in $L_{\mapsto\!\!\!\dashv}$ have safety conditions such as $e \in [n_1, n_2]$ where $n_1, n_2 \in \mathbb{Z}$. In this setting, transitions from locations in $L_{\mapsto}$ or $L_{\leftrightarrow}$ to locations in $L_{\mapsto\!\!\!\dashv}$ are considered as *back-edge* too. Since taking back-edge transitions while synchronizing from a subset S of locations is not avoidable, we can use bounded safety conditions to establish a reduction from halting problem in Minsky machines to provide the following undecidability result.

▶ **Lemma 8.** *The existence of a location-synchronizing word from a set S of locations in $\mathcal{A}$ under general safety condition* Safe *is undecidable.*

In the absence of back-edges, we get rid of bounded safety conditions, by explicitly encoding the energy values in locations at the expense of an exponential blowup. We thus assign non-negative safety condition to the encoded location and reduce to Lemma 6.

▶ **Lemma 9.** *Assume that $\mathcal{A}$ has no back-edge. The existence of a location-synchronizing word from $S \subseteq L_{\leftrightarrow}$ in $\mathcal{A}$ under general safety condition* Safe *is decidable in* 3-EXPSPACE, *and it is* EXPSPACE-*hard.*

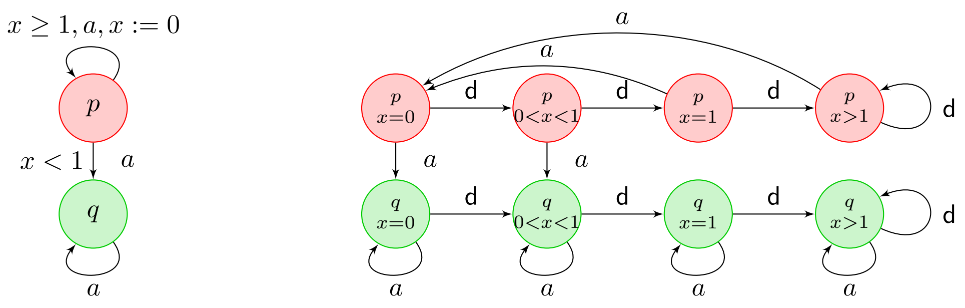
▶ **Theorem 10.** *The existence of a location-synchronizing word in a WA $\mathcal{A}$ with general safety condition* Safe *is decidable in* 3-EXPSPACE, *and it is* PSPACE-*hard.*

4 Synchronization in TAs

This section focuses on deciding the existence of a synchronizing and location-synchronizing word for TAs, proving PSPACE-completeness of the problems for deterministic TAs (without safety conditions, *i.e.*, no invariants), and proving undecidability for non-deterministic TAs.

4.1 Synchronization in deterministic TAs

We consider synchronizing words in TAs to be *timed words* that are sequences $w = a_0 a_1 \cdots a_n$ with $a_i \in \Sigma \cup \mathbb{R}_{\geq 0}$ for all $0 \leq i \leq n$. For a timed word, the *length* is the number of letters



■ **Figure 6** A TA and its region automaton (d is a special letter indicating delay transitions). The region automaton is synchronized by the word $a \cdot a \cdot \mathsf{d} \cdot \mathsf{d} \cdot \mathsf{d}$, but the TA cannot be synchronized (because there is no way to reset the clock when starting from location q).

in Σ it contains, and the *granularity* is infinite if the word involves non-rational delays, and it is the largest denominator if the timed word only involves rational delays.

We assume that the reader is familiar with the classical notion of region equivalence: this equivalence partitions the set of clock valuations into finitely many classes (called *regions*), and two states in the same location and region are time-abstract bisimilar. The *region automaton* is then a finite-state automaton obtained by quotienting the original TA with the region equivalence. We refer to [1] for a detailed presentation of this concept. The TA depicted in Fig. 6 exemplifies the fact that the region equivalence is not sound to find a synchronizing word. This is because region equivalence abstracts away the exact value of the clocks, while synchronizing needs to keep track of them.

To establish a PSPACE algorithm for deciding the existence of a synchronizing word for deterministic TAs, we first prove the existence of a *short witness* (in the sequel, a timed word is *short* when its length and granularity are in $O(2^{|C|} \times |L| \times |\mathcal{R}|)$). The built short witness starts with a *finitely-synchronizing* word, a word that brings the infinite set of states of the automaton to a finite set, and continues by synchronizing the states of this finite set pairwise.

▶ **Lemma 11.** *All synchronizing deterministic TAs have a short finitely-synchronizing word.*

Proof. We fix a complete deterministic TA $\mathcal{A} = \langle L, C, E \rangle$ with the maximal constant M. We begin with two folklore remarks on TAs. For all locations ℓ, we denote by $L_\ell = \{(\ell, v) \mid v(x) > M \text{ for all clocks } x \in C\}$ the set of states with location ℓ and where all clocks are unbounded; L_ℓ is one of the states in the region automata of $\mathcal{A}$.

▶ Remark. For all locations ℓ and for all timed words w, the set $\mathsf{loc}(\mathsf{post}(L_\ell, w))$ is a singleton and $\mathsf{post}(L_\ell, w)$ is included in a single region.

Notice that above Remark is a special property of L_ℓ, and in general: elapsing the same delay from two region-equivalent valuations may lead to non-equivalent valuations. The second remark is technical and provides the length and granularity of timed words that are needed for solving reachability in TAs.

▶ Remark. For all locations ℓ and all region r' such that (ℓ', r') is reachable from L_ℓ in the region automaton of $\mathcal{A}$, there exists a short timed word w of length at most $|L| \times |\mathcal{R}|$ (where $\mathcal{R}$ is the set of regions, whose size is exponential in the size of the automaton [1]) and two valuations $v \in r$ and $v' \in r'$ such that $\mathsf{post}((\ell, v), w) = \{(\ell', v')\}$.

Now, assuming that $\mathcal{A}$ has a synchronizing word, we build a short finitely-synchronizing w_f word with a key property: for all clocks $x \in C$, irrespective of the starting state, the run over w_f takes some transition resetting x. We first argue that for all clocks $x \in C$, from all states where $v(x) \neq 0$, there exists a reachable x-resetting transition. Towards contradiction, assume that there exist some state (ℓ, v) and clock x such that x will never be reset along any run from (ℓ, v). Runs starting from states with the same location ℓ but different clock valuations, say (ℓ, v') with $v'(x) \neq v(x)$, over a synchronizing word w, may either (1) reset x, and thus the final values of x on two runs from (ℓ, v) and (ℓ, v') are different, or (2) not reset x, so that the difference between $v(x)$ and $v'(x)$ is preserved along the runs over w. Both cases give contradiction, and thus for all clocks $x \in C$, from all states with $v(x) \neq 0$, there exists a reachable x-resetting transition.

Pick a valuation ℓ and a clock x. Applying the argument above to an arbitrary state of L_ℓ and clock x, we get a timed word $w_{\ell,x}$. By first Remark, inputting the same timed word from any state of L_ℓ always leads to the same transition resetting x. Moreover, all such runs end up in the same region. Note that by second Remark, $w_{\ell,x}$ can be chosen to have length and granularity at most $|L| \times |\mathcal{R}|$.

Below, we construct the short finitely synchronizing word w_f for $\mathcal{A}$ where S is the infinite set of states to be (finitely) synchronized (i.e., $\mathsf{post}(S, w_f)$ must be a finite set). Repeat the following procedure: pick a location ℓ such that there is an infinite set $S_\ell \subseteq S$ of states with the location ℓ in S. For each clock x, iteratively, input a word that consists of a $(M+1)$-time-unit delay and the word $w_{\ell,x}$. The timed word of $M+1$ delay brings the infinite set S_ℓ to the unbounded region L_ℓ. Next, following $w_{\ell,x}$ make the runs starting from S_ℓ end up in a single region where clock x has the same value for all runs (since it has been reset). The word $w_\ell = (\mathsf{d}(M+1) \cdot w_{\ell,x})_{x \in C}$ synchronizes the infinite set S_ℓ to a single state by resetting all clocks, one-by-one, and it also shrinks S. We repeat the procedure for next location $\ell' \in \mathsf{loc}(\mathsf{post}(S, w_\ell))$ until S is synchronized to a finite set. Note that for all locations ℓ, the word w_ℓ has length at most $|C| \times |L| \times (|\mathcal{R}| + 1)$ and granularity at most $|L| \times |\mathcal{R}|$. Thus the word w_f, obtained by concatenating the successive words w_ℓ, has length bounded by $|C| \times |L|^2 \times (|\mathcal{R}| + 1)$ and granularity at most $|L| \times |\mathcal{R}|$, so that it is short. By construction, it finitely-synchronizes $\mathcal{A}$, which concludes our proof. ◀

From the proof of Lemma 11, we see that for all synchronizing TAs, there exists a finitely-synchronizing word which, in a sense, synchronizes the clock valuations. Precisely:

▶ **Corollary 12.** *For all synchronizing deterministic TAs, there exists a short finitely-synchronizing word w_f such that for all locations ℓ, w_f synchronizes the set $\{\ell\} \times (C \to \mathbb{R}_{\geq 0})$ into a single state.*

Lemma 13 uses Corollary 12 to construct a short synchronizing word for a synchronizing TA. A short synchronizing word consists of a *finitely-synchronizing* word followed by a *pairwise synchronizing* word (i.e., a word that iteratively synchronizes pairs of states).

▶ **Lemma 13.** *All synchronizing deterministic TAs have a short synchronizing word.*

A naive algorithm for deciding the existence of a synchronizing word would consist in non-deterministically picking a short timed word, and checking whether it is synchronizing. However, the latter cannot be done easily, because we have infinitely many states to check, and the region automaton is not sound for this.

▶ **Theorem 14.** *Deciding the existence of a synchronizing word in a deterministic TA is* PSPACE*-complete.*

Proof. Given a complete deterministic TA $\mathcal{A}$ with the maximal constant M, we first consider the set $S_0 = \{(\ell, \mathbf{0}) \mid \ell \in L\}$ and compute the successors $\mathsf{post}(S_0, w_f)$ reached from S_0 by

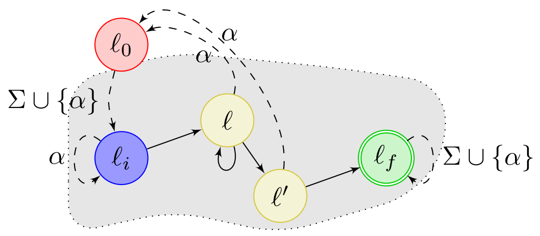


Figure 7 (Schematic) reduction from reachability to synchronizing word.

a finitely-synchronizing word w_f (built in the proof of Lemma 11). This can be achieved using polynomial space, since S_0 contains polynomially many states and w_f can be guessed on-the-fly. Moreover, since w_f begins with a delay of $M+1$ time unit, the set $\mathsf{post}(S_0, w_f)$ is equal to the set $\mathsf{post}(Q, w_f)$ where $Q = L \times \mathbb{R}^C_{\geq 0}$ is the state space of the semantic $[\![\mathcal{A}]\!]$ of the TA $\mathcal{A}$. The set $\mathsf{post}(S_0, w_f)$ contains at most $|L|$ states, which can now be synchronized pairwise. This phase can be achieved by computing the product automaton $\mathcal{A}^2$ and solving reachability problems in that automaton. This algorithm runs in polynomial space, and successfully terminates if, and only if, $\mathcal{A}$ has a synchronizing word.

The PSPACE-hardness proof is by a reduction from reachability in TA. The encoding is rather direct: given a deterministic TA $\mathcal{A}$ (w.l.o.g. we assume that $\mathcal{A}$ is complete) and two locations ℓ_i and ℓ_f, the existence of a run from $(\ell_i, \mathbf{0})$ to some state (ℓ_f, v) (with arbitrary v) is encoded as follows (see Fig. 7):

- add an extra letter α to the alphabet: $\Sigma \cup \{\alpha\}$;
- remove all outgoing edges from ℓ_f, and add a self-loop which is always available and resets all the clocks;
- add a self-loop on ℓ_i for α, which is always available and resets all the clocks;
- add a location ℓ_0, with a transition to ℓ_i which is always available and resets all clocks;
- for each location ℓ (except ℓ_0, ℓ_i and ℓ_f), add a transition $(\ell, \mathtt{true}, \alpha, C, \ell_0)$ to ℓ_0.

The resulting automaton $\mathcal{A}'$ is deterministic and complete.

▶ **Lemma 15.** *The automaton $\mathcal{A}'$ has a synchronizing word if, and only if, there exists some clock valuation v such that $\mathcal{A}$ has a run from $(\ell_i, \mathbf{0})$ to (ℓ_f, v).* ◀

Using similar arguments, we obtain the following result:

▶ **Theorem 16.** *Deciding the existence of a location-synchronizing word in a TA is* PSPACE-*complete.*

4.2 Synchronization in non-deterministic TAs

We now show the undecidability of the synchronizing-word problem for non-deterministic TAs. The proof is by a reduction from the non-universality problem of timed language for non-deterministic TAs, which is known to be undecidable [1].

▶ **Theorem 17.** *The existence of a (location-)synchronizing word in a non-deterministic TA is undecidable.*

Proof. Let $\mathcal{A} = \langle L, C, E\rangle$ be a non-deterministic TA over Σ, that we equip with an initial location ℓ_i and a set F of accepting locations (w.l.o.g. we assume that $\mathcal{A}$ is complete). From $\mathcal{A}$, we construct another TA $\mathcal{A}'$ over Σ' as follows (see Fig. 8):

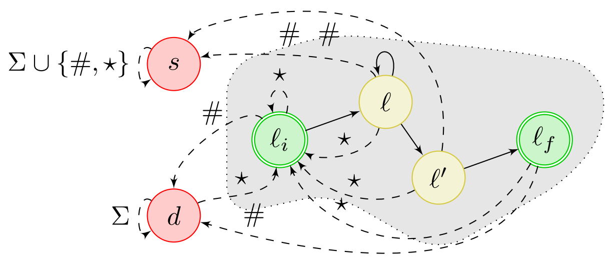


Figure 8 (Schematic) reduction from non-universality to synchronizing word (the newly added transitions are dashed; they all reset all the clocks. In this example: $\{\ell_i, \ell_f\} \subseteq F$.)

- the alphabet is augmented with two new letters $\#$ and $\star$.
- the set of locations of $\mathcal{A}'$ is $L \cup \{d, s\}$ (assuming $d, s \notin L$). Location s is a sink location, carrying a self-loop for all letters of the alphabet. Location d is a "departure" location: it also carries a self-loop for all letters, except for $\star$, which leads to ℓ_0. Those transitions all reset all the clocks.
- from all locations in L, there is a $\star$-transition to ℓ_i along which all the clocks are reset. From the states not in F, there is a $\#$-transition to s along which all clocks are reset. From the states in F, the $\#$-transition goes to d and reset all clocks.

▶ **Lemma 18.** *The language of $\mathcal{A}$ is not universal if, and only if, $\mathcal{A}'$ has a (location-) synchronizing word.*

The same reduction is used to show undecidability of the location-synchronizing problem; note that all transitions going to s (the only possible location to synchronize) always reset all clocks. Therefore, $\mathcal{A}'$ is synchronizing if, and only if, it is location synchronizing. By taking `true` safety condition for all locations (i.e., all states are safe), these two results also imply the undecidability of (location-)synchronizing problem with safety condition. ◀

References

1 Rajeev Alur and David L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
2 Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, and E. Shapiro. DNA molecule provides a computing machine with both data and fuel. *National Acad. Sci. USA*, 100:2191–2196, 2003.
3 Ján Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964.
4 L. Doyen, T. Massart, and M. Shirmohammadi. Infinite synchronizing words for probabilistic automata. In *Proc. of MFCS*, LNCS 6907, pages 278–289. Springer, 2011.
5 Laurent Doyen, Line Juhl, Kim G. Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing words for timed and weighted automata. Research Report LSV-13-15, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2013. 23 pages.
6 F. M. Fominykh and M. V. Volkov. P(l)aying for synchronization. In Nelma Moreira and Rogério Reis, editors, *CIAA'12*, volume 7381 of *LNCS*, pages 159–170. Springer, 2012.
7 P. V. Martyugin. Complexity of problems concerning carefully synchronizing words for PFA and directing words for NFA. In Farid M. Ablayev and Ernst W. Mayr, editors, *CSR'10*, volume 6072 of *LNCS*, pages 288–302. Springer, 2010.
8 M. V. Volkov. Synchronizing automata and the Černý conjecture. In *LATA'08*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.

Finite-Valued Weighted Automata*

Emmanuel Filiot[1], Raffaella Gentilini[2], and Jean-François Raskin[1]

1 Université Libre de Bruxelles
2 Universitá di Perugia

Abstract

Any weighted automaton (WA) defines a relation from finite words to values: given an input word, its set of values is obtained as the set of values computed by each accepting run on that word. A WA is k-valued if the relation it defines has degree at most k, i.e., every set of values associated with an input word has cardinality at most k. We investigate the class of quantitative languages defined by k-valued automata, for all parameters k. We consider several measures to associate values with runs: sum, discounted-sum, and more generally values in groups.

We define a general procedure which decides, given a bound k and a WA over a group, whether this automaton is k-valued. We also show that any k-valued WA over a group, under some general conditions, can be decomposed as a union of k unambiguous WA. While inclusion and equivalence are undecidable problems for arbitrary sum-automata, we show, based on this decomposition, that they are decidable for k-valued sum-automata, and k-valued discounted sum-automata over inverted integer discount factors. We finally show that the quantitative Church problem is undecidable for k-valued sum-automata, even given as finite unions of deterministic sum-automata.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Nested word, Transducer, Streaming

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.133

1 Introduction

Finite-state acceptor machines have found many applications in computer science. One of the most famous and studied example is the class of finite-state automata, which enjoys good algorithmic and closure properties. For instance, the decision problems of inclusion and equivalence, are decidable for finite-state automata. Finite-state automata and some of their variants have been successfully applied, for instance, to the theory of model-checking [6]. Their behaviour is however purely *Boolean* (either they accept their input or not). In many applications, this abstraction is not sufficient to accurately model systems where *quantitative* aspects are important. Weighted automata (WA) have been introduced to overcome this modelling weakness, as they can define *quantitative languages*, i.e. functions from words to values in some arbitrary set. WA have been studied for long [9] but recently, applications in computer-aided verification have been considered and new theoretical questions have been addressed [3]. In contrast with finite-state automata, the inclusion problem is however undecidable for some classes of WA, such as automata over the tropical semiring. One of the main goal of this paper is to recover decidability for some expressive classes of WA.

* This work is partially supported by FNRS (Mobilité 2014) and by INdAM-GNCS (Project 2014). We thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

Weighted automata (on finite words) extend finite-state automata with values which, in general, are taken in a semiring $(S, \oplus, \otimes)$. In this paper however, we focus on sum (weighted automata over the tropical semiring) and discounted sum automata, but generalise our results whenever it can be done. The value of an accepting run of a sum (resp. discounted sum) automaton A on a finite word is the (left-to-right) sum (resp. discounted sum for a discount factor $\lambda \in]0, 1[$) of all the values occurring along that run. The value $A(w)$ of a word w is defined only if there exists an accepting run on w, as the maximum of all the values of the accepting runs on w. As an example, consider the quantitative language L which associates with each word w over the alphabet $\{a, b\}$, the value $L(w) = max(\#_a(w), \#_b(w))$, where $\#_x(w)$ is the number of occurrences of x in w. This language L can be defined by a sum automaton A defined as the union of two disjoint deterministic WA A_a and A_b: For all $x \in \{0, 1\}$, A_x has a single (accepting and initial) state with a 1-weighted loop on reading x, and a 0-weighted loop on reading the other symbol.

The (quantitative) *inclusion problem* generalizes the classical inclusion problem for Boolean languages. It is the problem of deciding, given two WA A and B, whether for all words u, if u is accepted by A, then it is accepted by B and $A(u) \leq B(u)$. Even for the class of sum automata, this problem is know to be undecidable [15]. In [10], we have introduced the class of *functional WA*, i.e. WA such that every accepting runs on the same input word have same value. We have shown, for several measures (sum, discounted sum, and ratio), that this class is decidable, and has decidable inclusion problem.

Contributions. In this paper, we generalise the class of functional WA to k-valued WA i.e., WA such that, for all input words w, the set of values computed by all the accepting runs on w has cardinality at most k. For instance, the WA A defining the language L is k-valued for all $k \geq 2$. We show, for the measures sum and discounted sum (over inverted integer discount factor $1/n$), that they have decidable inclusion (and therefore decidable equivalence). Given $k \in \mathbb{N}$ and a sum-automaton (resp. a discounted-sum automaton) A, we prove that checking whether A is k-valued is decidable. We also show that if A is k-valued, then it can be decomposed as finite union of functional (i.e. 1-valued) sum-automata (resp. discounted-sum automata).

The last two results are more generally shown for *group automata (GA)*. Group automata extend finite state automata with weights over an *infinitary* group, i.e. a group $(G, \otimes)$ that satisfies some condition called the infinitary condition. This condition implies that two runs of a WA, on the same input, that synchronize on loop with different delays (i.e. the difference between their output values before and after taking the loop are non equal), can generate infinitely many different delays when iterating that loop. The semantics of a group automaton is a function from finite words w to the set of values (in G) of all the accepting runs on w. It is k-valued if any set of values associated with w has cardinality at most k. As we show, sum- and discounted-sum automata can be treated as group automata, as far as the max operation that combines the values of a word is not relevant. Real-time string-to-string transducers are also group automata (over the free group generated by the output alphabet), as they satisfy the infinitary condition. Therefore, our results (decidability of k-valuedness and decomposition) do not only apply to sum- and discounted-sum automata, but more generally to group automata, and as a particular case, we recover some results that were already known for string transducers. We now detail our contributions:

k-valuedness problem. First, we show that given $k \in \mathbb{N}$ and a GA A, checking whether A is k-valued is decidable. The proof of decidable k-valuedness for GA relies on a pumping argument that allows us to bound the size of non k-valuedness witnesses. Applied to our

measures, this general procedure for testing k-valuedness is in PSpace for both sum and discounted sum automata (seen as GA), when k is part of the input. The k-valuedness problem is shown to be PSpace-hard for sum automata, when k is part of the input. When k is fixed, we also show that testing k-valuedness can be done in PTime for sum automata, based on PTime complexity result for checking the existence of a zero-circuit in a multi-weighted graph [17]. Still for a fixed parameter k, we give another general k-valuedness checking procedure for GA which, applied to discounted sum, provides us with a PTime complexity. Applied to sum, this also yields PTime complexity with a worst constant than the ad-hoc PTime procedure. This general procedure extends to GA a procedure that was defined in [18] to decide k-valuedness of string-to-string transducers. Besides generalizing the procedure of [18], we show here how to simplify it and provide a simpler correctness proof based on the small witness property for non k-valuedness.

Decomposition of k-valued group automata. We show that any k-valued GA is equivalent to a union of k functional GA. Our decomposition technique non-trivially extends to GA a procedure that has been introduced for string-to-string transducers [19].

Inclusion, equivalence, and synthesis problems. Based on the decomposition result, we give a general scheme to decide inclusion of two k-valued sum (resp. discounted sum) WA. This scheme reduces the inclusion problem to checking the existence of a path with strictly positive sum (resp. discounted sum) on all dimensions of a multi-weighted graph. We show that this latter problem is decidable for sum (in PTime), and discounted sum (in PSpace) with inverted discount factors $1/n$, $n \in \mathbb{N}$. Therefore, it yields decidable inclusion and equivalence problems for these two classes of k-valued WA. If the two WA are given as unions of deterministic automata and known to have included domains, this procedure is PTime for sum automata, and PSpace for discounted sum automata. For general rational discount factors, the problem is still open and related to other open problems identified in, e.g., [4] and [2]. Finally, we consider the quantitative synthesis problem: the alphabet is partitioned into two sets that are controlled respectively by two players. The winning objective of the protagonist is given by a sum automaton A. The problem is to decide whether the protagonist has a strategy to choose his letters such that whatever letters the opponent chooses, the outcome of their interaction (which is a word) has strictly positive value by A. This problem was shown to be undecidable for functional sum-automata and decidable for deterministic sum-automata in [10]. Here, we show that it is undecidable for the union of p deterministic sum-automata, for $p \geq 4$.

Related Works. Comparisons with [10, 18, 19] have already been mentioned before. The notion of k-valuedness originates from the theory of string-to-string transducers. For general (non-deterministic) transducers, inclusion and equivalence are undecidable [11], but decidable for k-valued transducers [12, 21, 7]. This latter result has then been extended to tree transducers [20]. The k-valuedness problem for string-to-string transducers has been shown to be decidable in several papers [12, 18, 21, 8], in PTime when k is fixed. Any sum automaton A can be transformed into a string-to-string transducer T_A over a unary output alphabet, by adding m to all the weights of A (where m is the minimal weight occurring on the transitions of A). Then A is k-valued iff T_A is. While this encoding allows us to reuse existing results on string-to-string transducers, the complexity results are not optimal, because the weights of A are encoded in binary, and their translation into strings is unary. Moreover, there is no such encoding for discounted sum automata and therefore we rather give general procedures at the level of infinitary groups.

Sum-automata over $\mathbb{N}$ (called distance automata) have been considered in several papers, and known results on the distance problem (deciding whether there exists an upper bound on the value of every input word) are nicely summarised in [22]. In particular in [22], the class of finite-valued distance automata is considered. Finite-valuedness (deciding whether there exists k such that the distance automaton is k-valued) is shown to be decidable by a direct reduction to the string transducer case, and therefore the complexity bound is not optimal when the weights are encoded in binary. The finite-valuedness problem is not adressed in this paper. We leave it however as future work and it seems, again, that the technique of [19] would nicely generalise to infinitary groups.

Finitely ambiguous sum automata are sum automata such that there exists a bound b on the number of accepting runs on the same input word. They are known to be equivalent to union of unambiguous sum automata [13, 14, 19, 22]. The best bound on the size of the union (which matches exactly the degree of ambiguity of the automaton) is obtained in [19]. Our decomposition of k-valued WA as unions of unambiguous automata directly uses these results as an intermediate step. In [13], it is shown that the equivalence problem for finitely ambiguous sum-automata is decidable. There is however no precise complexity result.

Finally, let us mention that for the strictly positive discounted sum problem on multi-weighted graphs, if one requires the discounted sum to be greater than or equal to 0 on all the dimensions, then it corresponds to an open problem, which is at least as difficult as the exact value (open) problem in 1-dimensional graph (i.e. decide in a weighted graph, whether there exists a path with discounted sum exactly 0), as shown in [4].

2 Weighted Automata

Let W be a set (called weight-set) and Σ be a finite alphabet. An *automaton* over Σ and W is a tuple $A{=}(Q, q_I, F, \delta, \gamma)$ where Q is a finite set of states, F is a set of final states, $q_I \in Q$ is the initial state, $\delta \subseteq Q{\times}\Sigma{\times}Q$ is the transition relation, and $\gamma : \delta \to W$ is an edge-labelling function, mapping δ onto a weight-set W. A run ρ of A over a word $w = \sigma_0 \dots \sigma_n \in \Sigma^*$ is a sequence $\rho = q_0\sigma_0q_1 \dots \sigma_nq_{n+1}$ such that $q_0 = q_I$ and for all $i \in \{0, \dots, n\}$, $(q_i, \sigma_i, q_{i+1}) \in \delta$. We write $\rho : q_0 \xrightarrow{w|m} q_{n+1}$ to denote that ρ is a run on w starting at q_0 and ending in q_{n+1}, where $m = \gamma(q_0, \sigma_0, q_1) \cdot \gamma(q_1, \sigma_1, q_2) \cdot \dots \cdot \gamma(q_n, \sigma_n, q_{n+1})$. We write $|\rho|$ to denote the length of ρ, where $|\rho| = n + 1$. The run ρ is *accepting* if $q_{n+1} \in F$. A state q in $A = (Q, q_I, F, \delta, \gamma)$ is said *co-accessible* (resp. *accessible*) by some word $w \in \Sigma^*$ if there exists some run $\rho : q \xrightarrow{w|m} q_f$ for some $q_f \in F$ (resp. some run $\rho : q_I \xrightarrow{w|m} q$). If such a word exists, we say that q is co-accessible (resp. accessible). A is said to be *trim* if all its states are both accessible and co-accessible. It is well-known that an automaton can be trimmed in polynomial time [9].

A pair of states (q, q') is co-accessible if there exists a word w such that q and q' are co-accessible by w. The domain of A, denoted by $\mathrm{dom}(A)$, is defined as the set of words $w \in \Sigma^+$ on which there exists some accepting run of A. Note that (Q, q_I, F, δ) is a classical finite state automaton over Σ. We say that A is *deterministic* if (Q, q_I, F, δ) is deterministic. We say that A is *unambiguous* if (Q, q_I, F, δ) admits at most one accepting run for each word.

Give a run $\rho = q_0\sigma_0 \dots \sigma_nq_{n+1}$ and $0 \le \ell \le n + 1$, we define $\rho[\ell] = q_\ell$, $\rho[{:}\ell] = q_0\sigma_0 \dots \sigma_{\ell-1}q_\ell$ and $\rho[\ell{:}] = q_\ell\sigma_\ell \dots q_{n+1}$. Given a run $\rho' = q'_0\sigma_0 \dots \sigma_nq'_{n+1}$ on the same input as ρ, we define the *synchronized product* $\rho \otimes \rho'$ as the sequence $(q_0, q'_0)\sigma_0 \dots \sigma_n(q_{n+1}, q'_{n+1})$. A pair (ℓ_1, ℓ_2) such that $0 \le \ell_1 < \ell_2 \le n + 1$ is called a *cycle* (or loop) in ρ if $\rho[\ell_1] = \rho[\ell_2]$. The synchronized operator $\otimes$ can be inductively extended to sequences of runs as follows: $\rho_1 \otimes \dots \otimes \rho_m = (\rho_1 \otimes \dots \otimes \rho_{m-1}) \otimes \rho_m$. A cycle (ℓ_1, ℓ_2) in $\rho_1 \otimes \dots \otimes \rho_m$ is sometimes called a *synchronizing cycle* to emphasise that we consider product of runs.

If W is equipped with an operation $\cdot$, we define the *value* $V(\rho)$ of a run $\rho{=}q_0\sigma_0{\dots}\sigma_nq_{n+1}$

in $A{=}(Q, q_I, F, \delta, \gamma)$ as: $V(\rho) = \gamma(q_0, \sigma_0, q_1)\cdot\gamma(q_1, \sigma_1, q_2)\cdot\dots\cdot\gamma(q_n, \sigma_n, q_{n+1})$ if ρ is accepting and $V(\rho) = \perp$, otherwise[1]. The relation $R_A = \{(w, V(\rho)) \mid w \in \Sigma^+, \rho \text{ is a run of } A \text{ on } w\}$ is called[2] the *relation induced by* A. For all $k \in \mathbb{N}$, R_A is k-valued if for all words $w \in \Sigma^+$, we have $|\{v \mid (w, v) \in R_A, v \neq \perp\}| \leq k$. In this case we say that A is *k-valued* (and functional if $k = 1$).

In this paper we consider weight sets having the algebraic structure of a group $(W, \cdot, \mathbb{1})$. Recall that a group is a structure $(S, \cdot, \mathbb{1})$, where S is a set, $\cdot : S \times S \to S$ is an associative operation, $\mathbb{1} \in S$ is a two sided identity element for $\cdot$ over S, and each element $s \in S$ admits an inverse $s^{-1} \in S$, such that $s^{-1} \cdot s = s \cdot s^{-1} = \mathbb{1}$ (the inverse is unique).

Functional automata over groups where studied in [10], where a polynomial functionality test was given based on the notion of *delay* between two runs [5, 9] (cfr. Definition 1). Moreover, [10] provided a determinization test for automata over *infinitary groups*, i.e. groups satisfying the *infinitary condition* (c.f.r. Definition 2, below). Intuitively, the infinitary condition ensures that iterating two runs on a parallel loop induces infinitely many delays.

▶ **Definition 1** (Delay). Let $A = (Q, q_I, F, \delta, \gamma)$ be an automaton over a weight-set group $(W, \cdot, \mathbb{1})$. Given two elements $d_1, d_2 \in W$, the *delay* between d_1 and d_2 is defined by $\mathsf{delay}(d_1, d_2) = d_1^{-1} \cdot d_2$. Let $p, q \in Q$. A value $d \in W$ is a *delay* for (p, q) if A admits two runs $\rho : q_I \xrightarrow{w} p$, $\rho' : q_I \xrightarrow{w} q$ on some $w \in \Sigma^*$ s.t. $\mathsf{delay}(\rho, \rho') {=}_{def} \mathsf{delay}(V(\rho), V(\rho')) = d$.

▶ **Definition 2** (Infinitary Condition). A group $(W, \cdot, \mathbb{1})$ is said *infinitary* if it satisfies the *infinitary condition* stating that for all $v_1, w_1, v_2, w_2 \in W$, if $v_1^{-1} w_1 \neq v_2^{-1} v_1^{-1} w_1 w_2$, then:

$$|\{v_2^{-h} v_1^{-1} w_1 w_2^h \mid h \geq 0\}| = \infty$$

A *group automaton (GA)* over Σ, is an automaton over Σ and an infinitary group. A *weighted automaton (WA)* over Σ is an automaton over Σ and a semiring (we refer the reader to [9] for a definition of semiring). If A is a weighted automaton over Σ and a semiring $(W, \oplus, \cdot)$, the *quantitative language* L_A defined by A is a mapping $L_A : \Sigma^* \to W \cup \{\perp\}$, where $\perp \notin W$, defined for all $w \in \Sigma^*$ by $L_A(w) = \perp$ if $w \notin \mathrm{dom}(A)$, and $L_A(w) = \bigoplus R_A(w)$ otherwise. Note that any weighted automaton over a semiring $(W, \oplus, \cdot)$ such that can be seen as a group automaton, provided $(W, \cdot, \mathbb{1})$ is an infinitary group, where $\mathbb{1}$ is the neutral element for $\cdot$.

Remarkable subclasses of WA, such as e.g. Sum- and Dsum-automata [3], can be seen as group automata, as outlined in the following remark (cfr. also Proposition 1).

▶ Remark (Sum- and Dsum-automata). Sum- (resp. Dsum-) automata are usually defined as WA $A = (Q, q_I, F, \delta, \gamma)$, where the weight set $\gamma : \delta \to \mathbb{Z}$ maps edges onto integers and the value of a run ρ is simply the *sum* (resp. *discounted sum*) of the weights along its edges. More precisely, given a run $\rho = q_0\sigma_0 q_1 \dots \sigma_n q_{n+1}$ in a Sum-automaton A, $V(\rho)$ is defined by $V(\rho) = \sum_{i=0}^{i=n} \gamma(q_i, \sigma_i, q_{i+1})$ if ρ is accepting, and by $V(\rho) = \perp$ otherwise. If A is a Dsum-automaton with discount factor $\lambda \in]0, 1[$, then $V(\rho)$ is defined by $\sum_{i=0}^{i=n} \lambda^i \gamma(q_i, \sigma_i, q_{i+1})$ if ρ is accepting, and by $\perp$ otherwise (we assume that $\perp < m$ for all $m \in \mathbb{Z}$).

Clearly, Sum-automata can be seen as group automata over the group $(\mathbb{Z}, +, 0)$. For Dsum-automata, consider the group $(W, \cdot, \mathbb{1})$, where $W = \mathbb{Q}^2$, $\cdot$ is defined by $(a, x) \cdot (b, y) = (\frac{1}{y} a +$

[1] Here $\perp \notin W$ is a fresh symbol used to represent the fact that the value of ρ undefined.
[2] As in [3], we do not consider the empty word as our weighted automata do not have initial and final weight functions. This eases our presentation but all our results carry over to the more general setting with initial and final weight function [9].

$b, xy)$, $\mathbb{1} = (0,1)$, and given $(a,x) \in W$, the inverse $(a,x)^{-1}$ is given by $(a,x)^{-1} = (-xa, x^{-1})$. Given $\lambda \in \mathbb{Q} \cap]0,1[$, a Dsum-automaton A on Σ can be seen as a group automaton over $(W, \cdot, \mathbb{1})$, by replacing each weight a in A with the pair $(a, \lambda), a \in \mathbb{Z}$. Let $w = w_0 \dots w_n \in \Sigma$, and consider a run $\rho : q_0 \xrightarrow{w} q_{n+1}$ on w in A. Then, ρ is valued by the pair $(a,x) = (\frac{1}{\lambda^n}\gamma(q_0, w_0, q_1) + \cdots + \gamma(q_n, w_n, q_{n+1}), \lambda^{n+1})$. Hence, (a,x) codes the value $\frac{ax}{\lambda} = \mathsf{Dsum}(\rho)$. It can be shown that groups involved in the above definitions satisfy the infinitary condition.

▶ Proposition 1. Sum *and* Dsum *automata can be coded as group automata (over an infinitary group).*

3 The k-Valuedness Problem

In this section, we consider the problem of determining whether a group automaton (for fixed and unfixed k). As a first result, we provide a pumping argument that can be turned into a simple k-valuedness test.

▶ **Lemma 3** (Pumping). *Let $A = (Q, q_I, F, \delta, \gamma)$ be a group automaton over $(W.\cdot, \mathbb{1})$. A is not k-valued iff there exists a word $w \in \Sigma^*$ having length $|w| \le |Q|^{k+1} + |Q|^{k+1} \cdot (\frac{k(k-1)}{2})^2$ such that A admits $k+1$ accepting runs with pairwise distinct values on w.*

Lemma 3 yields immediately a decidability procedure for deciding if a given group automaton is k-valued: It is sufficient to guess $k+1$ runs of bounded length and check if they are accepting and the outputs are pairwise distinct. Such a procedure turns out to be PSPACE if the accumulated values along the guessed runs can be stored using polynomial space. In particular, this is the case for the classes of Sum- and Dsum-automata. Therefore,

▶ **Theorem 4.** *The k-valuedness problem is decidable for the class of group automata. When k is part of the input, it is PSPACE-complete on the subclass of* Sum*-automata and PSPACE-easy on the subclass of* Dsum*-automata.*

▶ Remark. Lemma 3 can be put in parallel with Theorem 3 in [8], establishing that k-valuedness for string to string transducers can be decided by analizing the image of a finite set of words, namely those having length $|Q|^{k+1}F(k)$, where $F(k)$ in [8] is an exponential function expressed in terms of a recurrence relation.

3.1 The k-Valuedness Problem when k is a Fixed Constant

In this subsection, we study the k-valuedness problem on group automata assuming k as a fixed constant. This assumption does not lower the complexity of the pumping based procedure of the latter section (for both measures), however in this section we give other, new procedures which provide better upper bounds for a fixed k. These new procedures, for a non-fixed k, have worse complexities than the pumping based procedure (Exptime).

The Case of Sum-Automata. Theorem 6 below shows that Sum-automata can be tested for k-valuedness in PTIME for fixed k. The polynomial k-valuedness algorithm designed within Theorem 6 relies onto a reduction to the problem of finding a strictly positive path (from the initial state to a final state) in a multi-weighted graph.

A k-weighted graph $G = (Q, q_I, E, F, w)$ is simply a weighted graph, where Q is the set states, q_I is the inital state, F is the set of final states, $E \subseteq Q \times Q$ is the edge relation and $w : E \mapsto \mathbb{Z}^k$ assigns to each edge a vector of integers of dimension k. A path $\pi = q_0, \dots, q_n$ in G is strictly positive if and only if $\sum_{i=0}^{n-1} w(q_i, q_{i+1}) > 0_{\mathbb{Z}^k}$. Lemma 5 below shows that deciding whether a k-weighted graph G contains a strictly positive path to a final state can be done in polynomial time.

▶ **Lemma 5.** *Determining if a k-weighted graph contains a strictly positive path from the inital state to a final state can be done in polynomial time.*

Intuitively, the multi-weighted graph G built in Theorem 6 to decide if a given Sum-automaton A is k-valued has dimension k. The vertices of G are $(k+1)$-tuples of states of A and G admits an edge e from $(q_1, \dots, q_{k+1})$ to $(p_1, \dots, p_{k+1})$ if and only if there exists $a \in \Sigma$ such that for all $1 \leq i \leq k+1$ $q_i \xrightarrow{a|n_i} p_i$ is an edge in A. Moreover the weight $w(e)$ of e is given by $(n_2 - n_1, n_3 - n_2, \dots, n_{k+1} - n_k)$. Therefore, A admits $k+1$ accepting runs $q_I \xrightarrow{w|v_1} p_1, \dots, q_I \xrightarrow{w|v_{k+1}} p_{k+1}$ on $w \in \Sigma^*$ with pairwise distinct values $v_1 < v_2 < \cdots < v_{k+1}$ if and only if the associated multi-weighted graph G admits a strictly positive path to $(p_1, \dots, p_{k+1})$.

▶ **Theorem 6.** *The k-valuedness problem on* Sum*-automata can be solved in polynomial time, when k is a fixed constant.*

▶ Remark. Note that for string transducers the k-valuedness problem can be reduced in polynomial time to the emptiness problem on one reversal nondeterministic multicounter machines[3] [12]. Although a similar reduction could be easily designed for Sum-automata, the overall procedure for testing k-valuedness would be only pseudopolynomial, as it turns out that emptiness of multicounter one-reversal machines where counters can be incremented by arbitrary constants is NP-complete.

The Case of Group Automata. In this subsection we show that the whole class of group automata can be tested for k-valuedness in polynomial time, assuming that k is an input fixed constant. The main ingredients of the polynomial test designed are the same as the ones used for the k-valuedness test on string transducers in [18], generalized from strings to groups. However, the pumping result stated in the previous section allows to significatively simplify the overall procedure of [18]. The first ingredient that we will use is the notion of *pairwise delays*, that is a natural extension of the concept of delay between two runs at the core of the functionality tests in [18]

▶ **Definition 7** (k-Pairwise Delay)**.** Let $(W, \cdot, \mathbb{1})$ be an infinitary group, let $k \in \mathbb{N}$. A k-pairwise delay on W is a function $\delta : \{\langle i, j\rangle \mid 1 \leq i < j \leq k\} \to W$.

We denote by $\Delta_k(\mathcal{G})$ the set of all k-pairwise delays on $\mathcal{G} = (W, \cdot, \mathbb{1})$ (omitting $\mathcal{G}$ and/or k when the latter are clear from the context).

▶ **Definition 8.** A group automaton $A = (Q, q_I, F, \delta, \gamma)$ on $(W, \cdot, \mathbb{1})$ admits a k-pairwise delay δ on $(q_1, \dots, q_k) \in Q^k$ iff there exists $w \in \Sigma^*$ such that for each pair $\langle i, j\rangle, 1 \leq i < j \leq k$, A admits two runs $q_I \xrightarrow{w|n} q_i$, $q_I \xrightarrow{w|m} q_j$ such that $n^{-1} \cdot m = \delta(\langle i, j\rangle)$. In this case, we say that w witnesses δ on $(q_1, \dots, q_k)$.

Given $A = (Q, q_I, F, \delta, \gamma)$ on $\mathcal{G} = (W, \cdot, \mathbb{1})$, and $k, i \in \mathbb{N}$, we define the function $D^i_k : Q^k \mapsto 2^{\Delta_k(\mathcal{G})}$ where $D^i_k(q_1, \dots, q_k)$ is the set of k-pairwise delay on $(q_1, \dots, q_k)$ witnessed by some word w of length $|w| \leq i$. Our pumping result ensures the existence of a bound b, polynomial in $|Q|$, such that D^b_{k+1} retains enough information to decide whether A is k-valued:

[3] The emptiness problem of reversal-bounded multicounter machines can be solved in polynomial time [12].

▶ **Lemma 9.** *Let $A = (Q, q_I, F, \delta, \gamma)$ be a group automaton over $(W, \cdot, \not\Vdash)$, let $b = |Q|^{k+1} + |Q|^{k+1} \cdot (\frac{k(k-1)}{2})^2$. A is not k-valued iff it admits a tuple of final states $(q_1, \dots, q_{k+1}) \in F^{k+1}$ such that $D^b_{k+1}(q_1, \dots, q_{k+1})$ contains a $(k+1)$-pairwise delay δ satisfying the following condition: for all $0 \leq i < j \leq k+1$, $\delta(\langle i, j\rangle) \neq \not\Vdash$.*

Unfortunately, given $(q_1, \dots, q_{k+1}) \in Q^{k+1}$, $D^b_{k+1}(q_1, \dots, q_{k+1})$ could contain exponentially many delays w.r.t. b, since each word w admits $\mathcal{O}(2^{|w|})$ runs on it. However, such delays can be properly abstracted using a generalization from strings to groups of the very powerful and elegant notion of *minimal traverse*, introduced in [18] to give a compact representation of the delays generated by a k-valued string transducer. Formally,

▶ **Definition 10** (k-Pairwise Partial Delay)**.** Let $(W, \cdot, \not\Vdash)$ be an infinitary group, let $k \in \mathbb{N}$. A k-pairwise partial delay on W is a function $\delta : \{\langle i, j\rangle \mid 1 \leq i < j \leq k\} \to W \cup \perp$.

We denote by $\Delta^\perp_k(\mathcal{G})$ the set of all k-pairwise partial delays on $\mathcal{G} = (W, \cdot, \not\Vdash)$. Given $\tau, \tau' \in \Delta^\perp_k(\mathcal{G})$, we say that τ is smaller (i.e. "more abstract") than τ' ($\tau \sqsubseteq \tau'$) if and only if for all pairs $1 \leq i < j \leq k$, if $\tau(\langle i, j\rangle) \neq \perp$ then $\tau(\langle i, j\rangle) = \tau'(\langle i, j\rangle)$. Note that the lattice defined by $\sqsubseteq$ on $\Delta^\perp_k(\mathcal{G})$ is not complete, namely τ, τ' admit a least upper bound if and only if they are compatible on the defined components.

Definition 11, below, formally introduces the notion of minimal traverse for a set of k-pairwise delays D. Intuitively, a minimal traverse τ for a set of k-pairwise delays D is so called for two reasons. First, it is a k-pairwise partial delay τ that traverses i.e. *intersects* each delay $\delta \in D$ on a pair of components (that is, $\tau(\langle i, j\rangle) = \delta(\langle i, j\rangle)$ for some $1 \leq i < j \leq k$). Second, it is minimal w.r.t. $\sqsubseteq$.

▶ **Definition 11** (Minimal Traverse)**.** Let D be a set of k-pairwise delays on $(W, \cdot, \not\Vdash)$. A *traverse* for D is a k-pairwise partial delay $\tau \in \Delta^\perp_k(\mathcal{G})$ such that:
- for each $\delta \in D$, there exists a pair $\langle i, j\rangle, 1 \leq i < j \leq k$ such that $\delta(\langle i, j\rangle) = \tau(\langle i, j\rangle) \neq \perp$
- for each pair $\langle i, j\rangle, 1 \leq i < j \leq k$, if $\tau(\langle i, j\rangle) \neq \perp$, then there exists $\delta \in D$ such that $\delta(\langle i, j\rangle) = \tau(\langle i, j\rangle)$.

A *minimal traverse* for D is a traverse for D minimal w.r.t. $\sqsubseteq$.

The set of minimal traverses for (a set of delays) D can obviously be empty, but more importantly its *maximum size depends only on* k, as stated in Lemma 12 (i.e. it is a constant, if k is assumed to be a constant). Given a set of delays D, we denote by $\alpha(D)$ the set of minimal traverses for D.

▶ **Lemma 12** ([18])**.** *For each set of k-pairwise delays D on $(W, \cdot, \not\Vdash)$, $|\alpha(D)| \leq 2^{k^4}$.*

We are now ready to present the announced abstraction for D^i_k. Given $D^i_k : Q^k \mapsto 2^{\Delta_k(\mathcal{G})}$, we denote by $\alpha(D^i_k) : Q^k \mapsto 2^{\Delta^\perp_k(\mathcal{G})}$ the function that associates with each tuple of states $(q_1, \dots, q_k) \in Q^k$, the set of minimal traverses $\alpha(D^i_k(q_1, \dots, q_k))$ for $D^i_k(q_1, \dots, q_k)$.

Lemma 13 below states that the abstraction α retains enough information to decide whether A is k-valued by inspecting $\alpha(D^b_{k+1})$, where $b = |Q|^{k+1} + |Q|^{k+1} \cdot (\frac{k(k-1)}{2})^2$ is the bound given by our pumping lemma.

▶ **Lemma 13.** *Let $A = (Q, q_I, F, \delta, \gamma)$ be a group automaton over $(W, \cdot, \not\Vdash)$, let $b = |Q|^{k+1} + |Q|^{k+1} \cdot (\frac{k(k-1)}{2})^2$. A is not k-valued iff it admits a tuple of final states $(q_1, \dots, q_{k+1}) \in F^{k+1}$ such that $\alpha(D^b_{k+1})(q_1, \dots, q_{k+1})$ does not contain a minimal traverse τ satisfying the following condition: $\tau$is equal to one on each pair of components on which it is not $\perp$, i.e. $\tau \sqsubseteq 1^{k+1}$.*

Lemma 14 states that $\alpha(D^{i+1}_k)$ can be computed from $\alpha(D^i_k)$ (in PTIME).

▶ **Lemma 14.** *Let $A = (Q, q_I, F, \delta, \gamma)$ be a group automaton. For all $i > 0$, $\alpha(D_k^i)$ can be computed in polynomial time wrt $|Q|$ and the size of $\alpha(D_k^{i-1})$.*

We are finally ready to state our polynomial result for testing k-valuedness:

▶ **Theorem 15.** *Let A be a group automaton over $(W, \cdot, \mathbb{1})$. If the delays accumulated along each polynomially bounded path of A can be computed in polynomial time, then k-valuedness can be tested in polynomial time for A, when k is a fixed constant.*

▶ **Corollary 16.** *The k-valuedness problem on* Dsum*-automata can be solved in polynomial time, when k is a fixed constant.*

4 Decomposition of k-valued group automata

In this section, we show that any k-valued group automaton A (over an infinitary group) is equivalent to a union of unambiguous group automata. Beside providing more insights toward expressivity issues, the equivalence established throughout this section will be used in Section 5 to provide positive results w.r.t. the decidability of the quantitative language inclusion problem on k-valued WA. The proof of this equivalence goes in two steps. First, we prove that A is equivalent to a k-ambiguous group automaton (for every input string, there are at most k accepting runs). The construction generalizes to groups that of [19] established for string-to-string transducers. Then, it is know that any k-ambiguous group automaton is equivalent to a union of unambiguous ones. This result has been proved in [19] for string transducers, based on a notion of lexicographic covering of k-ambiguous automata that can be directly applied to group automata.

From k-valued to k-ambiguous group automata. We assume that $G = (W, \cdot, \mathbb{1})$ is a group that satisfies the infinitary condition. Recall that $\mathsf{delay}(u, v) = u^{-1} \cdot v$ for all $u, v \in W$. The construction we present in this section generalizes to groups the construction of [19], proved for string-to-string transducers. In [19], the k-ambiguous equivalent transducer non-deterministically guesses, when it reads an input string u, an output string v and a run ρ that produces v such that all the runs on u that are strictly smaller than ρ (according to a lexicographic order on runs) either (i) produce a different value, or (ii) have a delay that exceeds some bound N on some prefix of u. For a sufficiently large N (that depends on the transducer only), there are at most k such runs ρ if the transducer is k-valued.

In order to check properties (i) and (ii), it suffices to store the delays between the current chosen run ρ with all the smaller runs. This is done by keeping track of all the states q the smaller runs can reach, together with all the possible delays associated with q. If some delay exceeds the bound N, it can safely be replaced by the value ∞, and therefore one only needs to store delays of length N at most. The main difficulty, when generalising this construction to groups, is that for groups, there is no notion of "long" delay since in general, the group is not equipped with a metric. Instead, the bound N we consider is on the number of different delays between the prefixes of the current chosen run ρ and the prefixes of the smaller runs. Given two runs ρ, ρ' on the same input word $u \in \Sigma^*$, we denote by $\mathsf{lag}(\rho, \rho')$ the set $\mathsf{lag}(\rho, \rho') = \{\mathsf{delay}(\rho[:i], \rho'[:i]) \mid 0 \leq i \leq |u| + 1\}$. Our construction is based on the following key lemma, that generalizes over group a similar lemma for strings proved in [19].

▶ **Lemma 17.** *Let A be a trim group automaton over G with n states. If A is k-valued, then for all tuples of runs $(\rho_1, \dots, \rho_{k+1})$ on the same input, there exists $1 \leq i < j \leq k+1$ such that:*
1. *ρ_i and ρ_j have the same output, i.e. $V(\rho_i) = V(\rho_j)$.*
2. *$|\mathsf{lag}(\rho_i, \rho_j)| \leq n^{k+1}$.*

Based on Lemma 17, we show that any k-valued trim GA $A = (Q, q_I, F, \Delta, \gamma)$ over G is equivalent to some k-ambiguous GA $A' = (Q', q'_I, F', \Delta', \gamma')$. We let $N = n^{k+1}$, where n is the number of states of A. The idea is to order the transitions of A with a total order $\prec_A$ and to extend it to a lexicographic order over runs. Then, the construction non-deterministically guesses a run ρ of A, and checks that all the runs that are smaller than ρ produce either a different value than ρ, or have a lag with ρ whose size exceeds N. Lemma 17 will ensure that there are at most k such runs ρ on the same input. In order to check these properties, A' guesses the transitions of ρ and for each state $q \in Q$, it stores all the lags (which are sets of delays) between the current prefix of ρ and the runs that end-up in q and are smaller than this prefix. If the size of a lag exceeds N, then A' replaces this lag by the extra value ∞. We will see that doing so, any lag that has to be stored by A' is a lag that can be generated by two runs of length n^{2k+3} at most, and therefore the state space will be finite.

Given a lag between two runs ρ_1 and ρ_2 and two transitions on the same symbol that extend these two runs respectively, in order to compute the lag of the extended runs, A' also needs to know the delay between ρ_1 and ρ_2. Therefore, A' also needs to store, for each lag, the "current" delay contained in this lag.

Construction of A'. Let us now define formally the state set Q' of A'. We let $\mathcal{D}$ be the set of all delays that can be generated by pairs of runs of length at most $N^2 n^4$ (we will show later why we need to take such a value). Note that $\mathcal{D}$ is a finite set. Let also $\mathcal{P}_N(\mathcal{D})$ be all the subsets of $\mathcal{D}$ of cardinality at most N. Then, we let $Q' = Q \times (Q \to 2^{(\mathcal{P}_N(\mathcal{D})\times\mathcal{D})\cup\{\infty\}})$. After reading an input string $u \in \Sigma^*$, A' is in state $(q, \ell) \in Q'$ iff q is the state of the current guessed run ρ of A on u, and for all $p \in Q$, $(L, d) \in \ell(p)$ iff there exists a run ρ' on u ending in p, smaller than ρ, and such that $\mathsf{lag}(\rho, \rho') = L$, $|L| \leq N$, and $\mathsf{delay}(\rho, \rho') = d$. Moreover, $\infty \in \ell(p)$ iff there exists a run ρ' on u ending in p, smaller than ρ, such that $|\mathsf{lag}(\rho, \rho')| > N$.

We accept a run iff it ends in a state (q, ℓ) such that q is accepting in A ($q \in F$), and there is no state $p \in Q$ such that $(L, \mathbb{1}) \in \ell(p)$ for some L (otherwise it would mean that there exists a smaller run which gives the same output than the guessed run, on the same input). So, $F' = \{(q, \ell) \in Q' \mid \forall p \in Q, \forall (L, d) \in \ell(p),\ d \neq \mathbb{1}\}$. The initial state of A' is $q'_I = (q_I, \ell_0)$ where q_I is the initial state of A, and for all $q \in Q$, $\ell_0(q) = \varnothing$ if $q \neq q_I$ and $\{(\{\mathbb{1}\}, \mathbb{1})\}$ if $q = q_I$. The transitions in Δ' are defined by: $(p, \ell) \xrightarrow{a|u} (p', \ell') \in \Delta'$ iff $p \xrightarrow{a|u} p' \in \Delta$ and for all $q' \in Q$, $\ell'(q') = \mathsf{prune}(\mathsf{update}(\ell, p, p', a, u, q'))$ where $\mathsf{update}(\ell, p, p', a, u, q')$ is the union of

$$\begin{array}{lc} L_1 & \{(L \cup \{u^{-1} \cdot d \cdot v\}, u^{-1} \cdot d \cdot v) \mid \exists q.(L, d) \in \ell(q) \wedge q \xrightarrow{a|v} q'\}, \\ L_2 & \{(\{u^{-1} \cdot v\}, u^{-1} \cdot v) \mid \exists p \xrightarrow{a|v} q' \prec_A p \xrightarrow{a|u} p'\}, and, \\ L_3 & \{\infty \mid \exists q.\infty \in \ell(q) \wedge \exists q \xrightarrow{a|v} q' \in \Delta\} \end{array} \quad .$$

and prune replaces each element (L, d) by ∞ if $|L| > N$, and $\mathsf{prune}(\infty) = \infty$.

It is easy to show that A' is well-defined, in the sense that the transition function is defined over states of Q' only, i.e. all the delays computed by A' can be generated by pairs of runs of length at most $N^2 n^4$.

▶ **Lemma 18.** *A and A' are equivalent. Moreover, A' is k-ambiguous.*

On the ground of the decomposition of k-valued GA as k-ambiguous GA, and the known decomposition of k-ambiguous automata into union of unambiguous automata, one gets finally the following equivalence theorem:

▶ **Theorem 19.** *Any k-valued GA over G is equivalent to the union of k unambiguous GA over G.*

5 The Inclusion and Synthesis Problems for k-Valued WA

Inclusion problem. In this section we consider k-valued V-automata, $V \in \{\mathsf{Sum}, \mathsf{Dsum}\}$ and we provide positive decidability results w.r.t. their inclusion problem. Given two V-automata on the alphabet Σ, the inclusion problem ($A \le B$?) asks whether for all $w \in \Sigma^+$, $L_A(w) \le L_B(w)$. This problem is undecidable for general Sum-automata, open for general Dsum-automata and PSPACE-c for functional Sum- and Dsum-automata [3, 10]. Relying on the results in the previous sections, we provide an inclusion test applying to k-valued Sum and to the class of Dsum-automata having a discount factor λ that is an inverted integer, i.e. we consider discount factors of the form $\frac{1}{n}, n \in \mathbb{N}^*$. In more details, we show how to encode the inclusion problem for Sum-automata (resp. Dsum-automata) into the problem of determining a strictly positive path (resp. a path with strictly positive discounted sum) in a multi-weighted graph.

Let A, B be two k-valued weighted V-automata, $V \in \{\mathsf{Sum}, \mathsf{Dsum}\}$. The encoding proceeds in three steps. First, we check whether $\mathrm{dom}(A) \subseteq \mathrm{dom}(B)$. Then, by Theorem 19, A and B can be effectively decomposed as unions of k unambiguous V-automata $\bigcup_{i=1}^{k} G_i$ and $\bigcup_{i=1}^{k} H_i$ respectively. We can assume that $dom(G_i) = dom(A)$ for all i (resp. $dom(H_i) = dom(B)$ for all i)[4]. Clearly, $A \not\le B$ iff there exist $w \in \Sigma^*$ and $i \in \{1, \dots, k\}$ such that $G_i(w) > B(w)$. For all $i \in \{1, \dots, k\}$, we finally check whether $G_I(w) > B(w)$ for some w (i.e. $G_i(w) > H_j(w)$ for some w and all j) by a reduction to a multi-dimensional graph problem, as follows.

Let $i \in \{1, \dots, k\}$, Q_i be the set of states of G_i, and P_j the set of states of H_j for all $j \in \{1, \dots, k\}$. We construct a k-weighted graph $\Phi_i = (V_i, q_i^0, F_i, E_i, w_i)$ where the vertices are $V_i = Q_i \times P_1 \times \cdots \times P_k$, the weight function has type $w_i : E_i \to \mathbb{Z}^k$, and the weighted transitions are defined by $(q, p_1, \dots, p_k) \xrightarrow{(v_1,\dots,v_k)} (q', p'_1, \dots, p'_k) \in E_i$ iff $q \xrightarrow{a|m} q'$ is a transition of G_i and for all $j \in \{1, \dots, k\}$, $p_j \xrightarrow{a|v_j+m} p'_j$ is a transition of H_j. Final vertices (resp. the initial vertex) in Φ_i are those whose components are all final (resp. initial) states.

Then, the following holds: $A \not\le B$ if and only if there exists $i \in \{1, \dots, k\}$ and a path π to a final state in Φ_i with strictly positive sum (resp. stricly positive discounted sum) on all components. By Lemma 5, it is possible to check in polynomial time whether a multi-weighted graph of dimension k contains a strictly positive path to a target vertex. Therefore, the inclusion problem is decidable for k-valued Sum-automata.

Lemma 20 below provides a PSPACE procedure to check whether a multi-weighted graph has a path to a given target state with strictly positive *discounted* sum w.r.t. a discounted factor λ that is an inverted integer $\frac{1}{n}, n \in \mathbb{N}^*$. This leads to decidability of the inclusion also for the class of Dsum-automata, when the discount factor λ is an inverted integer[5].

▶ **Lemma 20.** *Let $(V, v_0, E, w : E \mapsto \mathbb{Z}^k)$ be a multi-weighted graph of dimension k, $t \in V$, and consider $\lambda = \frac{1}{n}, n \in \mathbb{N}^*$. The problem of deciding whether G admits a path from v_0 to t with strictly positive discounted sum on all components w.r.t. λ is in PSPACE.*

▶ **Theorem 21.** *The inclusion problem is decidable for k-valued* Sum*-automata and k-valued* Dsum*-automata associated with a discount factor λ such that $\lambda = \frac{1}{n}, n \in \mathbb{N}^*$.*

[4] Otherwise, we opportunely complete G_i.

[5] Note that inverted integer discounted factors where considered in [1] to provide a determinization procedure applying to *complete* Dsum-automata (all states are accepting). However, for Dsum-automata with final states no special form of discount factor can guarantee determinization [1].

▶ Remark. If A and B are given in input as union of k_1 and k_2 unambiguous WA with the same domain, then checking whether $A \leq B$ can be done in polynomial time (resp. PSPACE) for Sum-automata (resp. Dsum-automata w.r.t. inverted integer discount factors) in virtue of Lemma 20, Lemma 5 and Theorem 21. Finally, note that the encoding into WA over groups povided by Proposition 1 allows to associate a different discount factor with each edge of the given automaton. The results in Theorem 21 can be further generalized in this sense: In particular, Proposition 1 guarantees the correctness of the encoding into the problem of determining a path with strictly positive discounted sum in a multi-weighted graph. The proof of Lemma 20, solving the latter problem, can be easily generalized to deal with different inverted integer discount factors.

Quantitative Synthesis. We consider *quantitative realizability* problem. The realizability problem is better understood as a game between two players: "Player input" (the environment, also called Player I) and "Player output" (the controller, also called Player O). Player I (resp. Player O) controls the letters of a finite alphabet Σ_I (resp. Σ_O). We assume that $\Sigma_O \cap \Sigma_I = \varnothing$ and that Σ_O contains a special symbol # whose role is to stop the game. We let $\Sigma = \Sigma_O \cup \Sigma_I$. Formally, the realizability game is a turn-based game played on an arena defined by a weighted automaton $A = (Q = Q_O \uplus Q_I, q_0, F, \delta = \delta_I \cup \delta_O, \gamma)$, whose set of states is partitioned into two sets Q_O and Q_I, $\delta_O \subseteq Q_O \times \Sigma_O \times Q_I$, $\delta_I \subseteq Q_I \times \Sigma_I \times Q_O$, and such that $\mathrm{dom}(A) \subseteq (\Sigma \backslash \{\#\})^*\#$. Player O starts by giving an initial letter $o_0 \in \Sigma_O$, Player I responds providing a letter $i_0 \in \Sigma_I$, then Player O gives o_1 and Player I responds i_1, and so on. Player O has also the power to stop the game at any turn with the distinguishing symbol #. In this case, the game results in a finite word $(o_0 i_0)(o_1 i_1)\dots(o_j i_j)\# \in \Sigma^*$, otherwise the outcome of the game is an infinite word $(o_0 i_0)(o_1 i_1)\cdots \in \Sigma^\omega$.

The *quantitative realizability problem* asks whether Player O has the strategy $\lambda_O : (\Sigma_O \Sigma_I)^* \to \Sigma_O$ such that for all Player I's strategies $\lambda_I : \Sigma_O(\Sigma_I \Sigma_O)^* \to \Sigma_I$, the outcome of this two strategie, which is a finite word w, satisfies $A(w) > 0$, in which case we say that A is realizable. We refer the reader to [10] for formal definitions of strategies and outcomes. In [10], we have shown that the realizability problem is undecidable for functional Sum WA (and for other measures), while a positive decidability results applies to deterministic Sum WA (and for other measure). As a corollary, we know that the problem is undecidable for k-valued Sum WA. We will strengthen this negative result here and show that even if the problem is decidable for deterministic WA, it is undecidable for Sum WA defined as union of k deterministic automata, for $k \geq 4$. In particular, we show that the halting problem for deterministic 2-counter Minsky machines [16] can be reduced to the quantitative language realizability problem for the union of 4-deterministic Sum-automata.

▶ **Theorem 22.** *The realizability problem for a 4-valued* Sum*-automata defined as a union of 4 deterministic* Sum*-automata is undecidable.*

References

1 U. Boker and T. A. Henzinger. Determinizing discounted-sum automata. In *CSL*, pages 82–96, 2011.

2 V. Bruyère, N. Meunier, and J-F. Raskin. Secure equilibria in weighted games. In *LICS*, 2014. to appear.

3 K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log*, 11(4), 2010.

4 K. Chatterjee, V. Forejt, and D. Wojtczak. Multi-objective discounted reward verification in graphs and mdps. In *LPAR*, pages 228–242, 2013.

5 C. Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977.
6 E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
7 R. de Souza. On the decidability of the equivalence for k-valued transducers. In *Developments in Language Theory*, pages 252–263, 2008.
8 R. de Souza and N. Kobayashi. A combinatorial study of k-valued rational relations. *Journal of Automata, Languages and Combinatorics*, 3/4(13):207–231, 2008.
9 M. Droste, W. Kuich, and H. Vogler. *HandBook of Weighted Automata*. Springer, 2009.
10 E. Filiot, R. Gentilini, and J-F. Raskin. Quantitative languages defined by functional automata. In *CONCUR*, pages 132–146, 2012.
11 T. V. Griffiths. The unsolvability of the equivalence problem for -free nondeterministic generalized machines. *Journal of the ACM*, 1968.
12 E. Gurari and O. Ibarra. A note on finitely-valued and finitely ambiguous transducers. *Mathematical Systems Theory*, 16(1):61–66, 1983.
13 K. Hashiguchi, K. Ishiguro, and S. Jimbo. Decidability of the equivalence problem for finitely ambiguous finance automata. *IJAC*, 12(3):445, 2002.
14 I. Klimann, S. Lombardy, J. Mairesse, and C. Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *TCS*, 327(3):349–373, 2004.
15 D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Int. Journal of Algebra and Computation*, 4(3):405–425, 1994.
16 M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
17 K. S. Rao and G. Sullivan. Detecting cycles in dynamic graphs in polynomial time. In *Proc. STOC'88*, STOC, pages 398–406. ACM, 1988.
18 J. Sakarovitch and R. de Souza. On the decidability of bounded valuedness for transducers. In *Proc. MFCS 2008*, MFCS. Springer-Verlag, 2008.
19 J. Sakarovitch and R. de Souza. Lexicographic decomposition of k -valued transducers. *Theory Comput. Syst*, 47(3):758–785, 2010.
20 H. Seidl. Equivalence of finite-valued tree transducers is decidable. *Mathematical Systems Theory*, 27(4):285–346, 1994.
21 A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27(8):749–780, 1989.
22 A. Weber. Finite-valued distance automata. *TCS*, 134(1):225–251, 7 November 1994.

First-order Definable String Transformations

Emmanuel Filiot[1], **Shankara Narayanan Krishna**[2], **and Ashutosh Trivedi**[2]

1 **F.N.R.S. Research Associate, Université Libre de Bruxelles, Belgium**
`efiliot@ulb.ac.be`

2 **Department of Computer Science and Engineering, IIT Bombay, India**
`krishnas,trivedi@cse.iitb.ac.in`

Abstract

The connection between languages defined by computational models and logic for languages is well-studied. Monadic second-order logic and finite automata are shown to closely correspond to each-other for the languages of strings, trees, and partial-orders. Similar connections are shown for first-order logic and finite automata with certain aperiodicity restriction. Courcelle in 1994 proposed a way to use logic to define functions over structures where the output structure is defined using logical formulas interpreted over the input structure. Engelfriet and Hoogeboom discovered the corresponding "automata connection" by showing that two-way generalised sequential machines capture the class of monadic-second order definable transformations. Alur and Cerny further refined the result by proposing a one-way deterministic transducer model with string variables – called the streaming string transducers – to capture the same class of transformations. In this paper we establish a transducer-logic correspondence for Courcelle's first-order definable string transformations. We propose a new notion of transition monoid for streaming string transducers that involves structural properties of both underlying input automata and variable dependencies. By putting an aperiodicity restriction on the transition monoids, we define a class of streaming string transducers that captures exactly the class of first-order definable transformations.

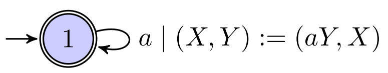
1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases First-order logic, streaming string transducers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.147

1 Introduction

The class of regular languages is among one of the most well-studied concept in the theory of formal languages. Regular languages have been precisely characterized widely by differing formalisms like monadic second-order logic (MSO), finite state automata, regular expressions, and finite monoids. The connection [8] between finite state automata and monadic second-order logic (MSO) is among the most celebrated results of formal language theory. Over the years, there has been substantial research to establish similar connections for the languages definable using first-order logic (FO) [11]. Aperiodic automata are restrictions of finite automata with certain aperiodicity restrictions defined through aperiodicity of their transition monoid. Recall that the transition monoid of an automaton A is the set of Boolean transition matrices M_s, for all strings s, indexed by states of A: $M_s[p][q] = 1$ if and only if there exists a run from p to q on s. The set of matrices M_s is a finite monoid. It is aperiodic if there exists $m \geq 0$ such that for all $s \in \Sigma^*$, $M_{s^m} = M_{s^{m+1}}$. Aperiodic automata define exactly FO-definable languages [17, 11]. Other formalisms capturing FO-definable languages include counter-free automata, star-free regular expressions, and very weak alternating automata.



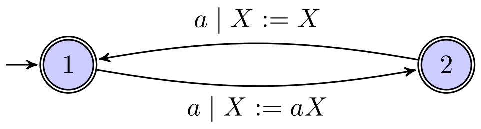


Figure 1 SSTs, T_0 (shown left) and T_1 (shown right), implement the transformation f_{halve}.

Beginning with the work of Courcelle [10], logic and automata connections have also been explored in context of string transformations. The first result in this direction is by Engelfriet and Hoogeboom [12], where MSO-definable transformations have been shown to be equivalent to two-way finite transducers. This result has then been extended to trees and macro-tree transducers [13]. Recently, Alur and Černý [1, 2] introduced *streaming string transducers*, a one-way finite transducer model extended with variables, and showed that they precisely capture MSO-definable transformations not only in finite string-to-string case, but also for infinite strings [6] and tree [3, 5] transformations.

Streaming string transducers (SSTs) manipulate a finite set of string variables to compute their output as they read the input string in one left-to-right pass. Instead of appending symbols to the output tape, SSTs concurrently update all string variables using a concatenation of output symbols and string variables in a *copyless* fashion, i.e. no variable occurs more than once in each concurrent variable update. The transformation of a string is then defined using an output (partial) function F that associates states with a copyless concatenation of string variables. It has been shown that SSTs have good algorithmic properties (such as decidable type-checking, equivalence) [1, 2] and naturally generalize to various settings like trees and nested words [3, 5], infinite strings [6], and quantitative languages [4].

In this paper we study FO-definable string transformation and recover a logic and transducer connection for such transformations. Such FO transformations, although weaker than MSO transformations, still enjoy a lot of expressive power: for instance they can still double, reverse, and swap strings, and are closed under FO look-ahead. We introduce a new concept of transition monoid for SST, used to define the notion of aperiodic SST to capture FO-definable transformations. To appreciate the challenges involved in finding the right definition of aperiodicity for SSTs consider the transformation f_{halve} defined as $a^n \mapsto a^{\lceil \frac{n}{2} \rceil}$ implemented by two SSTs shown in in Figure 1. Intuitively f_{halve} is not FO-definable since it requires to distinguish based on the parity of the input. Consider, the SST T_1 shown in Figure 1 with 2 accepting states and 1 variable.

Readers familiar with aperiodic automata may notice that the automata corresponding to T_1 is not aperiodic, but indeed has period 2. It seems a valid conjecture that SSTs whose transition monoid of underlying automaton is aperiodic characterize first-order definable transformations. However, unfortunately this is not a sufficient condition as shown by the SST T_0 in Figure 1 which also implements f_{halve} (its output is $F(1) = X$). In this example, although the underlying automaton is aperiodic, variables contribute to certain non aperiodicity.

We capture the notion of aperiodicity in SSTs by introducing the notion of variable flow. We say that by reading letter a, variable X flows to Y (if the update of variable Y is based on variable X). The notion of transition monoid is extended to SSTs to take both state and variable flow into account. We define transition matrices M_s indexed by pairs (p, X) where p is a state and X is a variable. Since in general, for copy-full SSTs, a variable X might be copied in more than one variable, it could be that X flows into Y several times. Our notion of transition monoid also takes into account, the number of times a variable flows into another. In particular, $M_s[p, X][q, Y] = i$ means that there exists a run from p to q on s on which X flows to Y for i number of times. Hence the transition monoid of an SST may not be finite. A key contribution of this paper is that FO string transformations are exactly the transformations definable by SSTs whose transition monoid is aperiodic with matrix values ranging over $\{0, 1\}$ (called 1-bounded transition monoid). In contrast with [1] our proof is not based on the intermediate model of two-way transducers and is more direct. We give a logic-based proof that simplifies that of [5] by restricting it to string-to-string transformations. We also show that checking aperiodicity of an SST is PSPACE-COMPLETE. Finally, simple restrictions on SST transition monoids naturally capture restrictions on variable updates that has been considered in other works. For instance, *bounded copy* of [6] correspond to finiteness of the transition monoid, while *restricted copy* of [3] correspond to its 1-boundedness.

Related work. Diekert and Gastin [11] presented a detailed survey of several automata, logical, and algebraic characterisations of first-order definable languages. As mentioned earlier the connection between MSO and transducers have been investigated in [1, 12]. A connection between two-way transducers and FO-transformations has been mentioned in [9] in an oral communication, where authors left the SST connection as an open question. First-order transformations are considered in [15], but not in the sense of [10]. In particular, they are weaker, as they cannot double strings or mirror them, and are definable by one-way (variable-free) finite state transducers. Finally, [7] considers first-order definable transformations *with origin information*. The semantics is different from ours, because these transformations are not just mapping from string to strings, but they also connect output symbols with input symbols from where they originate. The first-order definability problem for regular languages is known to be decidable. In particular, given a deterministic automaton A, deciding whether A defines a first-order language can be decided in PSPACE. Although we make an important and necessary step in answering this question in the context of regular string transformation, the decidability remains an open problem.

For the lack of space proofs are either sketched or omitted; full proofs can be found in [14].

2 Preliminaries

A string over a finite alphabet Σ is a finite sequence of letters from Σ. We write ϵ for the empty string and by Σ^* for the set of strings over Σ. A language over Σ is a subset of Σ^*. For a string $s \in \Sigma^*$ we write $|s|$ for its length and $\mathrm{dom}(s)$ for the set $\{1, \ldots, |s|\}$ of its positions. For all $i \in \mathrm{dom}(s)$ we write $s[i]$ for the i-th letter of the string s. For any $j \in \mathrm{dom}(s)$, the substring starting at position i and ending at position j is defined as ϵ if $j < i$ and by the sequence of letters $s[i]s[i+1] \ldots s[j]$ otherwise. We write $s[i{:}j]$, $s(i{:}j)$, $s[i{:}j)$, and $s(i{:}j]$, to denote substrings of s respectively starting i and ending at j, starting at $i{+}1$ and ending at $j{-}1$, and so on.

We represent a string $s \in \Sigma^*$ by the relational structure $\Xi_s{=}(\mathrm{dom}(s), \preceq^s, (L^s_a)_{a\in\Sigma})$, called the string model of s, where $\preceq^s$ is a binary relation over the positions in s characterizing the natural order, i.e. $(i, j) \in \preceq^s$ if $i \leq j$; L^s_a, for all $a \in \Sigma$, are the unary predicates that hold for the positions in s labeled with the alphabet a, i.e., $L^s_a(i)$ iff $s[i] = a$, for all $i \in \mathrm{dom}(s)$. When it is clear form context we drop the superscript s from the relations $\preceq^s$ and L^s_a.

2.1 First-order logic for strings

Properties of strings over Σ can be formalized by first-order logic denoted by FO(Σ). Formulas of FO(Σ) are built up from variables $x, y, \ldots$ ranging over positions of string models along with *atomic formulas* of the form $x{=}y, x{\preceq}y$, and $L_a(x)$ for all $a \in \Sigma$. Atomic formulas are connected with *propositional connectives* $\neg$, $\wedge$, $\vee$, $\rightarrow$, and *quantifiers* $\forall$ and $\exists$ that range over node variables. We say that a variable is *free* in a formula if it does not occur in the scope of some quantifier. A *sentence* is a formula with no free variables. We write $\phi(x_1, x_2, \ldots, x_k)$ to denote that at most the variables $x_1, \ldots, x_k$ occur free in ϕ. For a string $s \in \Sigma^*$ and for positions $n_1, n_2, \ldots, n_k \in \mathrm{dom}(s)$ we say that s with valuation $\nu = (n_1, n_2, \ldots, n_k)$ satisfies the formula $\phi(x_1, x_2, \ldots, x_k)$ and we write $(s, \nu) \models \phi(x_1, x_2, \ldots, x_k)$ or $s \models \phi(n_1, n_2, \ldots, n_k)$ if formula ϕ with n_i as the interpretations of x_i satisfies in string model Ξ_s. The language defined by an FO sentence ϕ is $L(\phi) \stackrel{\text{def}}{=} \{s \in \Sigma^* \ : \ \Xi_s \models \phi\}$. We say that a language L is FO-definable if there is an FO sentence ϕ such that $L = L(\phi)$.

2.2 Aperiodic Finite Automata

A finite automaton (FA)is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is an input alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, and $F \subseteq Q$ is the set of accepting states. (q, a, q') denotes a transition of the automaton $\mathcal{A}$ from q to q' on a; this is written as $q \xrightarrow{a} q'$. We write $q_0 \rightsquigarrow^s_{\mathcal{A}} q_n$ to denote a run from q_0 to q_n on string s; (or $q_0 \rightsquigarrow^s q_n$ if the automaton is clear from the context) s is accepted if $q_n \in F$. The language defined by a finite automaton $\mathcal{A}$ is $L(\mathcal{A}) = \{s \ : \ q_0 \rightsquigarrow^s q_n \text{ and } q_n \in F\}$.

Recall that a monoid is an algebraic structure $(M, \cdot, e)$ with a non-empty set M, a binary operation $\cdot$, and an identity element $e \in M$ such that for all $x, y, z \in M$ we have that $(x \cdot (y \cdot z)){=}((x \cdot y) \cdot z)$, and $x \cdot e = e \cdot x$ for all $x \in M$. We say that a monoid $(M, \cdot, e)$ is *finite* if the set M is finite. We say that a monoid $(M, ., e)$ is *aperiodic* [17] if there exists $n \in \mathbb{N}$ such that for all $x \in M$, $x^n = x^{n+1}$. Note that for finite monoids, it is equivalent to require that for all $x \in M$, there exists $n \in \mathbb{N}$ such that $x^n = x^{n+1}$. The following monoids are of special importance in this paper.

1. **Free Monoid.** The set of all strings over Σ, denoted as $(\Sigma^*, ., \epsilon)$ and known as the free monoid, has string concatenation as the operation and the empty string ϵ as the identity.
2. **Transition Monoid.** The set of transition matrices of an automaton $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ forms a finite monoid with matrix multiplication as the operation and the unit matrix $\mathbf{1}$ as the identity element. This monoid is denoted as $\mathcal{M}_{\mathcal{A}} = (M_{\mathcal{A}}, \times, \mathbf{1})$ and known as transition monoid of $\mathcal{A}$. Formally, the set $M_{\mathcal{A}}$ is the set of $|Q|$-square Boolean matrices $M_A = \{M_s \ : \ s \in \Sigma^*\}$ where for all strings $s \in \Sigma^*$, we have that $M_s[p][q] = 1$ iff $p \rightsquigarrow^s q$.

We say that a FA is aperiodic if its transition monoid is aperiodic. It is well-known [17] that a language $L \subseteq \Sigma^*$ is FO-definable iff it is accepted by some aperiodic FA.

3 Aperiodic String Transducers

For sets A and B, we write $[A \rightarrow B]$ for the set of functions $F : A \rightarrow B$, and $[A \rightharpoonup B]$ for the set of partial functions $F : A \rightharpoonup B$. A string-to-string transformation from an input alphabet Σ to an output alphabet Γ is a partial function in $[\Sigma^* \rightharpoonup \Gamma^*]$. We have seen some examples of string-to-string transformations in the introduction. For the examples of first-order definable transformations we use the following representative example.

▶ **Example 1.** Let $\Sigma{=}\{a, b\}$. For all strings $s \in \Sigma^*$, we denote by $\overline{s}$ its mirror image, and for all $\sigma \in \Sigma$, by $s \backslash \sigma$ the string obtained by removing all symbols σ from s. The transformation $f_1 : \Sigma^* \rightharpoonup \Sigma^*$ maps any string $s \in \Sigma^*$ to the output string $(s \backslash b)\overline{s}(s \backslash a)$. For example, $f_1(abaa) = aaa.aaba.b$.

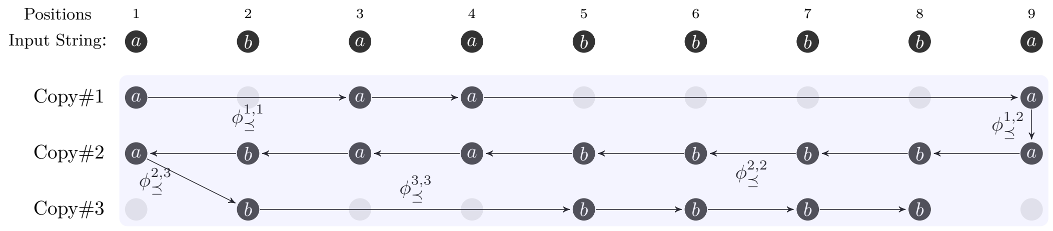


Figure 2 First-Order Transduction $w \mapsto (w\backslash b)\overline{w}(w\backslash a)$.

3.1 First-order logic definable Transformations

Courcelle [10] initiated the study of structure transformations using monadic second-order logic. In this paper, we restrict this logic-based transformation model to FO-definable string transformations. The main idea of Courcelle's transformations is to define a transformation $(w, w') \in R$ by defining the string model of w' using a finite number of copies of positions of the string model of w. The existence of positions, various edges, and position labels are then given as FO(Σ) formulas.

An *FO string transducer* is a tuple $T{=}(\Sigma, \Gamma, \phi_{\mathrm{dom}}, C, \phi_{\mathrm{pos}}, \phi_{\preceq})$ where: Σ and Γ are (finite) input and output alphabets; ϕ_{dom} is a closed FO(Σ) formula characterizing the domain of the transformation; $C{=}\{1, 2, \dots, n\}$ is a finite index set; $\phi_{\mathrm{pos}}{=}\big\{\phi^c_\gamma(x) : c \in C \text{ and } \gamma \in \Gamma\big\}$ is a finite set of FO(Σ) formulas with a free position variable x; $\phi_{\preceq}{=}\Big\{\phi^{c,d}_{\preceq}(x, y) : c, d \in C\Big\}$ is a finite set of FO(Σ) formulas with two free position variables x and y. The transformation $[\![T]\!]$ defined by T is as follows. A string s with $\Xi_s = (\mathrm{dom}(s), \preceq, (L_a)_{a\in\Sigma})$ is in the domain of $[\![T]\!]$ if $s \models \phi_{\mathrm{dom}}$ and the output is the relational structure $M = (D, \preceq^M, (L^M_\gamma)_{\gamma\in\Gamma})$ such that $D = \{v^c \ : \ c \in \mathrm{dom}(s), c \in C \text{ and } \phi^c(v)\}$ is the set of positions where $\phi^c(v) \overset{\mathrm{def}}{=} \vee_{\gamma\in\Gamma}\phi^c_\gamma(v)$; $\preceq^M \subseteq D{\times}D$ is the ordering relation between positions and it is such that for $v, u \in dom(s)$ and $c, d \in C$ we have that $v^c \preceq^M u^d$ if $w \models \phi^{c,d}_{\preceq}(v, u)$; and for all $v^c \in D$ we have that $L^M_\gamma(v^c)$ iff $\phi^c_\gamma(v)$. Observe that the output is unique and therefore FO transducers implement functions. However, note that the output structure may not always be a string. We say that an FO transducer is a *string-to-string* transducer if its domain is restricted to string graphs and the output is also a string graph.

We say that a string-to-string transformation is FO-definable if there exists an FO string-to-string transducer implementing the transformation and write FOT for the set of FO-definable string-to-string transformations. We define the *quantifier rank* $qr(T)$ of an FOT T as the maximal quantifier rank of any formula in T, plus 1. We add 1 for technical reasons, mainly because defining the successor relation requires one quantifier.

▶ **Example 2.** Consider the transformation f_1 of Example 1. It can be defined using an FO transducer that uses three copies of the input domain as shown in Fig. 2.

The domain formula ϕ_{dom}, an FO formula, simply characterizes valid string models. The first copy corresponds to $(w\backslash b)$, therefore the label formula $\phi^1_\gamma(x)$ is defined by false if $\gamma = b$ in order to filter out the input positions labelled b, and by true otherwise. The second copy corresponds to $\overline{w}$, hence all positions of the input are kept and their labels preserved, but the edge direction is complemented; hence the label formula is $\phi^2_\gamma(x) = L_\gamma(x)$. The third copy corresponds to $(w\backslash a)$ and hence $\phi^3_\gamma(x)$ is true if $\gamma = b$ and false otherwise. The transitive closure of the output successor relation is defined by $\phi^{1,1}_{\preceq}(x, y) = x \preceq y$, $\phi^{2,2}_{\preceq}(x, y) = y \preceq x$, $\phi^{3,3}_{\preceq}(x, y) = x \preceq y$, $\phi^{c,c'}_{\preceq}(x, y) = \text{true if } c < c'$, and $\phi^{c,c'}_{\preceq}(x, y) = \text{false if } c' < c$. Note that the transitive closure is not depicted on the figure, but only the successor relation.

Using first-order logic we define the position successor relation the following way: for all copies c, d, the existence of a direct edge from a position x^c to a position y^d of the output, also called the successor relation $S(x^c, y^d)$, is defined by the formula $\phi_{\text{succ}}^{c,d}(x,y) \stackrel{\text{def}}{=} \phi_{\prec}^{c,d}(x,y) \land \neg\exists z. \bigvee_{e \in C} \phi_{\prec}^{c,e}(x,z) \land \phi_{\prec}^{e,d}(z,y)$ where $\phi_{\prec}^{c_1,c_2}(x_1,x_2) \stackrel{\text{def}}{=} \phi_{\preceq}^{c_1,c_2}(x_1,x_2) \land x_1 \neq x_2$ for all $c_1, c_2 \in C$.

3.2 Streaming String Transducers

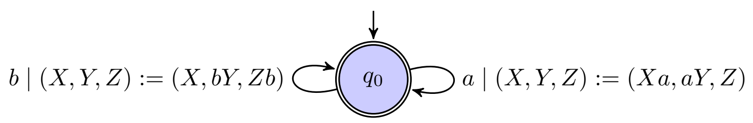
Streaming string transducers [1, 2] are one-way finite-state transducers that manipulates a finite set of string variables to compute its output. Instead of appending symbols to the output tape, SSTs concurrently update all string variables using a concatenation of string variables and output symbols. The transformation of a string is then defined using an output (partial) function F that associates states with a concatenation of string variables, s.t. if the state q is reached after reading the string and $F(q){=}xy$, then the output string is the final valuation of x concatenated with that of y. In this section we formally introduce SSTs and introduce restrictions on SSTs that capture FO-definable transformations.

Let $\mathcal{X}$ be a finite set of variables and Γ be a finite alphabet. A substitution σ is defined as a mapping $\sigma : \mathcal{X} \to (\Gamma \cup \mathcal{X})^*$. A valuation is defined as a substitution $\sigma : \mathcal{X} \to \Gamma^*$. Let $\mathcal{S}_{\mathcal{X},\Gamma}$ be the set of all substitutions $[\mathcal{X} \to (\Gamma \cup \mathcal{X})^*]$. Any substitution σ can be extended to $\hat{\sigma} : (\Gamma \cup \mathcal{X})^* \to (\Gamma \cup \mathcal{X})^*$ in a straightforward manner. The composition $\sigma_1\sigma_2$ of two substitutions σ_1 and σ_2 is defined as the standard function composition $\hat{\sigma_1}\sigma_2$, i.e. $\hat{\sigma_1}\sigma_2(X) = \hat{\sigma_1}(\sigma_2(X))$ for all $X \in \mathcal{X}$. We now introduce streaming string transducers.

▶ **Definition 3.** A streaming string transducer is a tuple $(\Sigma, \Gamma, Q, q_0, Q_f, \delta, \mathcal{X}, \rho, F)$ where: (1) Σ and Γ are (finite) input and output alphabets; (2) Q is a finite set of states with initial state q_0; (3) $\delta : Q \times \Sigma \to Q$ is a transition function; (4) $\mathcal{X}$ is a finite set of variables; (5) $\rho : (Q \times \Sigma) \to \mathcal{S}_{\mathcal{X},\Gamma}$ is a variable update function; (6) Q_f is a subset of final states; and (7) $F : Q_f \rightharpoonup \mathcal{X}^*$ is an output function.

The concept of a run of an SST is defined in an analogous manner to that of a finite state automaton. The sequence $\langle \sigma_{r,i} \rangle_{0 \leq i \leq |r|}$ of substitutions induced by a run $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots q_{n-1} \xrightarrow{a_n} q_n$ is defined inductively as the following: $\sigma_{r,i}{=}\sigma_{r,i-1}\rho(q_{i-1}, a_i)$ for $1 < i \leq |r|$ and $\sigma_{r,1} = \rho(q_0, a_1)$. We denote $\sigma_{r,|r|}$ by σ_r. If the run r is final, i.e. $q_n \in Q_f$, we can extend the output function F to the run r by $F(r) = \sigma_\epsilon \sigma_r F(q_n)$, where σ_ϵ substitutes all variables by their initial value ϵ. For all strings $s \in \Sigma^*$, the output of s by T is defined only if there exists an accepting run r of T on s, and in that case the output is denoted by $T(s) = F(r)$. The transformation $[\![T]\!]$ defined by an SST T is the function $\{(s, T(s)) \,:\, T(s) \text{ is defined}\}$.

▶ **Example 4.** Let us consider the SST T_2 with one state q_0 and three variables X, Y, and Z, shown below implementing the transformation f_1 introduced in Example 1. The variable update is shown in the figure and the output function is s.t. $F(q_0) = XYZ$.


$b \mid (X,Y,Z) := (X, bY, Zb)$
q_0
$a \mid (X,Y,Z) := (Xa, aY, Z)$

Let r be the run of T_2 on $s = abaa$. We have $\sigma_{r,1} : (X,Y,Z) \mapsto (Xa, aY, Z)$, $\sigma_{r,2} : (X,Y,Z) \mapsto \sigma_{r,1}(X, bY, Zb) = (Xa, baY, Zb)$, $\sigma_{r,3} : (X,Y,Z) \mapsto \sigma_{r,2}(Xa, aY, Z) = (Xaa, abaY, Zb)$ and $\sigma_{r,4} : (X,Y,Z) \mapsto \sigma_{r,3}(Xa, aY, Z) = (Xaaa, aabaY, Zb)$. Therefore we have that

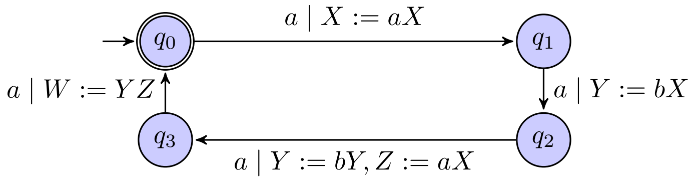
$$T(s) = F(r) = \sigma_\epsilon \sigma_{r,4} F(q_0) = \sigma_\epsilon \sigma_{r,4}(XYZ) = \sigma_\epsilon(XaaaaabaYZb) = aaaaabab.$$

3.3 SSTs: Transition Monoid and Aperiodicity

We define the notion of aperiodic SSTs by introducing an appropriate notion of transition monoid for transducers. The transition monoid of an SST T is based on the effect of a string s on the states and variables. The effect on variables is characterized by, what we call, flow information that is given as a relation that describes the number of copies of the content of a given variable that contribute to another variable after reading a string s.

State and Variable Flow. Let $T = (\Sigma, \Gamma, Q, q_0, Q_f, \delta, \mathcal{X}, \rho, F)$ be an SST. Let s be a string in Σ^* and suppose that there exists a run r of T on s. Recall that this run induces a substitution σ_r that maps each variable $X \in \mathcal{X}$ to a string $u \in (\Gamma \cup \mathcal{X})^*$. For string variables $X, Y \in \mathcal{X}$, states $p, q \in Q$, and $n \in \mathbb{N}$ we say that n copies of Y flow to X from p to q if there exists a run r on s from p to q, and Y occurs n times in $\sigma_r(X)$. We denote the flow with respect to a string s as $(p, Y) \rightsquigarrow_n^s (q, X)$.

▶ **Example 5.** Consider the run r from q_0 to q_0 over the string $aaaa$ in the following SST. To minimize clutter, while drawing SSTs we omit updates of variables that remain unchanged.



On the run r on $aaaa$, $\sigma_{r,4}(W) = \sigma_{r,3}[W := YZ] = \sigma_{r,3}(Y)\sigma_{r,3}(Z)$. However, $\sigma_{r,3}(Y) = b\sigma_{r,2}(Y) = b.b.\sigma_{r,1}(X)$ and $\sigma_{r,3}(Z) = a.\sigma_{r,2}(X) = a.\sigma_{r,1}(X)$, and $\sigma_{r,1}(X) = a$. Now for run r we have $(q_0, X) \rightsquigarrow_2^{aaaa} (q_0, W)$.

In order to define the transition monoid of an SST T, we first extend $\mathbb{N}$ with an extra element $\bot$, and let $\mathbb{N}_\bot = \mathbb{N} \cup \{\bot\}$. This new element behaves as 0, i.e. for all $i \in \mathbb{N}_\bot$, $i.\bot = \bot.i = \bot$, $i + \bot = \bot + i = i$. Moreover, we assume that $\bot < n$ for all $n \in \mathbb{N}$. We assume that pairs $(p, X) \in Q \times \mathcal{X}$ are totally ordered.

▶ **Definition 6** (Transition Monoid of SSTs). The *transition monoid* of a streaming string transducer T is the set of square matrices over $\mathbb{N}_\bot$ indexed (in order) by elements of $Q \times \mathcal{X}$, defined by $M_T = \{M_s \mid s \in \Sigma^*\}$ where for all strings $s \in \Sigma^*$, $M_s[p, Y][q, X] = n \in \mathbb{N}$ if and only if $(p, Y) \rightsquigarrow_n^s (q, X)$, and $M_s[p, Y][q, X] = \bot$ if and only if there is no run from p to q on s. By definition, there is atmost one run r from (p, Y) to (q, X) on any string s.

It is easy to see that $(M_T, \times, \mathbf{1})$ is a monoid, where $\times$ is defined as matrix multiplication and the identity element is the unit matrix $\mathbf{1}$. The mapping $M_\bullet$, which maps any string s to its transition matrix M_s, is a morphism from $(\Sigma^*, ., \epsilon)$ to $(M_T, \times, \mathbf{1})$. We say that the transition monoid M_T of an SST T is n-bounded if all the coefficients of the matrices of M_T are bounded by n. Clearly, any n-bounded transition monoid is finite.

In the original definition [2] of SST, updates were *copyless*, i.e., the content of a variable can never flow into two different variables, and cannot flow more than once into another variable. In [3], this condition was slightly relaxed to the notion of *restricted copy*, where a variable cannot flow more than once into another variable. This allows for a limited form of copy: for instance, X can flow to Y and Z, but Y and Z cannot flow to the same variable. Finally, *bounded copy* SSTs were introduced in [6] as a restriction on the variable dependency graphs. This restriction requires that there exists a bound K such that any variable flows at most K times in another variable. These three restrictions were shown to be equivalent, in the sense that SSTs with copyless, restricted copy, and bounded copy updates have the same expressive power. Due to our definition of transition monoid, and the results of [6], Theorem 7 is immediate by observing that bounded copy restriction of [6] for SSTs corresponds to finiteness of transition monoid. Also, notice that since the bounded copy assumption generalizes the copyless [2] and restricted copy [3] assumptions, previous definitions in the literature of SSTs correspond to finite transition monoids.

▶ **Theorem 7** ([6]). *[MSO-definable string transformations] A string transformation is MSO-definable iff it is definable by an* SST *with finite transition monoid.*

The main goal of this paper is to present a similar result for FO-definable transformations. For this reason we define aperiodic and 1-bounded SSTs.

▶ **Definition 8** (Aperiodic and 1-bounded SSTs). An SST is aperiodic if its transition monoid is aperiodic. An SST is 1-bounded if its transition monoid is 1-bounded, i.e. for all strings s, and all pairs (p, Y), (q, X), $M_s[p, Y][q, X] \in \{\perp, 0, 1\}$. See [14] for an example.

It can be shown (see [14]) that the domain of an aperiodic SST is FO-definable. We show that an SST is non-aperiodic iff its transition monoid contains a non-trivial cycle. Checking the existence of a non-trivial cycle is in PSpace for deterministic automata [16].

▶ **Lemma 9.** *Checking aperiodicity and 1-boundedness for SSTs is* PSpace-complete.

Now we are in a position to present the main result of this paper. We prove the following key theorem using Lemma 15 (Section 5) and Lemma 11 (Section 4).

▶ **Theorem 10** (FO-definable string transformations). *A string transformation is FO-definable iff it is definable by an aperiodic, 1-bounded* SST.

4 From aperiodic 1-bounded SST to FOT

▶ **Lemma 11.** *A string transformation is FO-definable* ***if it is*** *definable by an aperiodic,1-bounded SST.*

The idea closely follows the SST-to-MSOT construction of [1, 6]. The main challenge here is to show that aperiodicity and 1-boundedness on the SST implies FO-definability of the output string structure (in particular the predicate $\preceq$). We first show that the variable flow of any aperiodic,1-bounded SST is FO-definable. This will be crucial to show that the output predicate $\preceq$ is FO-definable.

Let $X \in \mathcal{X}$, $s \in \mathrm{dom}(T)$, $i \in \mathrm{dom}(s)$, and let $n = |s|$. We say that the pair (X, i) is *useful* if the content of variable X before reading $s[i]$ will be part of the output after reading the whole string s. Formally, if $r = q_0 \dots q_n$ is the accepting run of T on s, then (X, i) is useful for s if $(q_{i-1}, X) \rightsquigarrow_1^{s[i:n]} (q_n, Y)$ for some variable $Y \in F(q_n)$. Thanks to the FO-definability of variable flow this property is FO-definable.

Next, we define the SST-output structure given an input string structure. It is an intermediate representation of the output, and the transformation of any input string into its SST-output structure will be shown to be FO-definable. For any SST T and string $s \in \mathrm{dom}(T)$, the SST-output structure of s is a relational structure $G_T(s)$ obtained by taking, for each variable $X \in \mathcal{X}$, two copies of $\mathrm{dom}(s)$, respectively denoted by X^{in} and X^{out}. For notational convenience we assume that these structures are labeled on the edges. This structure satisfies the following invariants: for all $i \in dom(s)$, (1) the nodes (X^{in}, i) and (X^{out}, i) exist only if (X, i) is useful, and (2) there is a directed path from (X^{in}, i) to (X^{out}, i) whose sequence of labels is equal to the value of the variable X computed by T after reading $s[i]$. The condition on usefulness of nodes implies that SST-output structures consist of a single directed component, and therefore they are edge-labeled string structures.

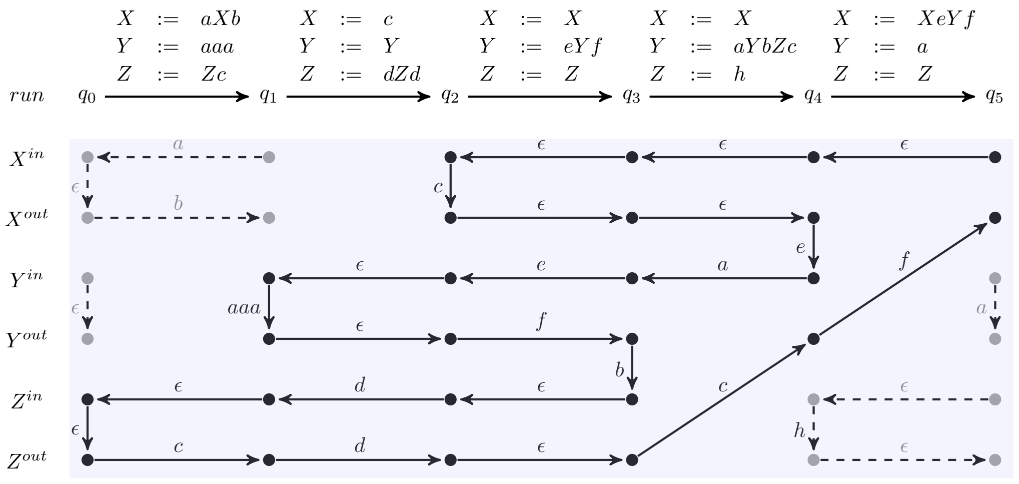


Figure 3 SST-output structure.

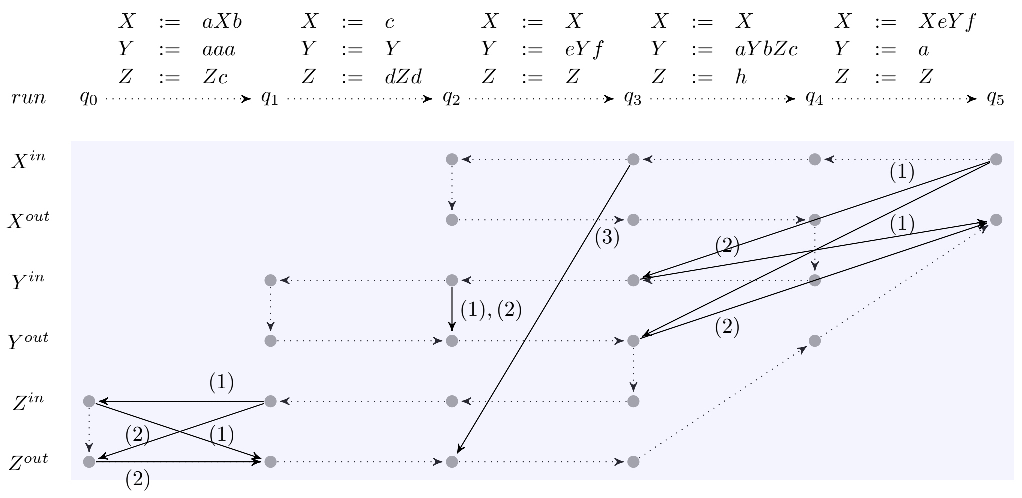
▶ **Example 12.** An example of SST-output structure in shown in Figure 3. Here we show only the variable updates. Dashed arrows represent variable updates for useless variables, and do not belong to SST-output structure; solid edges belong to the SST-output structure. Initially the variable content of Z is ϵ, this is represented by the ϵ-edge from $(Z^{in}, 0)$ to $(Z^{out}, 0)$ in the first column. Then, variable Z is updated to Zc. Hence, the new content of Z starts with ϵ (represented by the ϵ-edge from $(Z^{in}, 1)$ to $(Z^{in}, 0)$, which is concatenated with the previous content of Z, and then concatenated with c (it is represented by the c-edge from $(Z^{out}, 0)$ to $(Z^{out}, 1)$). The output is given by the path from $(X^{in}, 5)$ to $(X^{out}, 5)$ and equals $ceaeaaafbdcdcf$. Also note that some edges are labelled by strings with several letters, but there are finitely many possible such strings. In particular, we denote by O_T the set of all strings that appear in right-hand side of variable updates.

What remains for us, is to adapt from [1, 6], the MSO-definability of the transformation that maps a string s to its SST-output structure : we show that it is FO-definable as long as the SST is aperiodic. The main challenge is to define the transitive closure of the edge relation in first-order. Let T be $(Q, q_0, \Sigma, \Gamma, \mathcal{X}, \delta, \rho, Q_f)$. The SST-output structures of T, as node-labeled strings, can be seen as logical structures over the signature $S_{O_T} = \{(E_\gamma)_{\gamma \in O_T}, \preceq\}$ where the symbols E_γ are binary predicates interpreted as edges labeled by O_T. We let E denote the edge relation, disregarding the labels. To prove that transitive closure is FO[Σ]-definable, we use the fact that variable flow is $FO[\Sigma]$-definable. The following property, along with the FO-definability of variable flow, shows that transitive closure is FO-definable.

▶ Proposition 1. Let T be an **aperiodic** SST T. Let $s \in dom(T)$, $G_T(s)$ its SST-output structure and $r = q_0 \dots q_n$ the accepting run of T on s. For all variables $X, Y \in \mathcal{X}$, all positions $i, j \in \text{dom}(s) \cup \{0\}$, all $d, d' \in \{in, out\}$, there exists a path from node (X^d, i) to node $(Y^{d'}, j)$ in $G_T(s)$ iff (X, i) and (Y, j) are both useful and one of the following holds:

1. $Y \rightsquigarrow^{r[j:i]} X$ and $d = in$,
2. $X \rightsquigarrow^{r[i:j]} Y$ and $d' = out$, or
3. there exists $k \geq max(i, j)$ and two variables X', Y' such $X \rightsquigarrow^{r[i:k]} X'$, $Y \rightsquigarrow^{r[j:k]} Y'$ and X' and Y' are concatenated in this order[1] by r when reading $s[k+1]$.

[1] By “concatenated” we mean that there exists a variable update whose rhs is of the form $\dots X' \dots Y' \dots$



■ **Figure 4** Conditions of Proposition 1.

▶ **Example 13.** We illustrate proposition 1 using example of Fig.4. We have for instance $(q_2, Y) \rightsquigarrow_1^{s[3:2]=\epsilon} (q_2, Y)$, therefore by conditions (1) (and (2)) by taking $X = Y$ and $i = j = 2$, there exists a path from $(Y^{in}, 2)$ to $(Y^{out}, 2)$. Note that none of these conditions imply the existence of an edge from $(Y^{out}, 2)$ to $(Y^{in}, 2)$, but self-loops on $(Y^{in}, 2)$ and $(Y^{out}, 2)$ are implied by conditions (1) and (2) respectively. Now consider positions 0 and 1 and variable Z. It is the case that $(q_0, Z) \rightsquigarrow_1^{s[1:1]} (q_1, Z)$, therefore by condition (1) there is a path from $(Z^{in}, 1)$ to $(Z^{in}, 0)$ and to $(Z^{out}, 0)$. Similarly, by condition (2) there is a path from $(Z^{in}, 0)$ to $(Z^{out}, 1)$ and from $(Z^{out}, 0)$ to $(Z^{out}, 1)$. For positions 3 and 5, note that $(q_3, Y) \rightsquigarrow_1^{s[4:5]} (q_5, X)$, hence there is a path from $(Y^d, 3)$ to $(X^{out}, 5)$ for all $d \in \{in, out\}$. By condition (2) one also gets edges from $(X^{in}, 5)$ to $(Y^d, 3)$. Finally consider nodes $(Z^{out}, 2)$ and $(X^{in}, 3)$. There is no flow relation between variable Z at position 2 and variable X at position 3. However, $(q_3, X) \rightsquigarrow_1^{s[4:4]} (q_4, X)$ and $(q_2, Z) \rightsquigarrow^{s[3:4]} (q_4, Y)$. Then X and Y gets concatenated at position 4 to define X at position 5. Hence, there is a path from $(X^{in}, 3)$ to $(Z^{out}, 2)$ (condition (3)).

▶ **Lemma 14.** *For an **aperiodic** SST T, variables $X, Y \in \mathcal{X}$ and all $d, d' \in \{in, out\}$, there exists an FO[Σ]-formula $path_{X,Y,d,d'}(x, y)$ with two free variables s.t. for all $s \in \mathrm{dom}(T)$ and $i, j \in dom(s)$, $s \models path_{X,Y,d,d'}(i, j)$ iff there exists a path from (X^d, i) to (Y^d, j) in $G_T(s)$.*

We now sketch the proof of Lemma 11. Let Γ be the output alphabet. We adapt the MSO-definability of strings to SST-output structures from [6, 1] and use the FO-definability of transitive closure (Lemma 14) to show that strings to SST-output structure transformations are FO-definable whenever the SST is aperiodic. Since the usefulness of nodes is FO-definable, we filter out useless nodes in the first FO-transformation, unlike [6, 1], where useless nodes in the SST-output structures are later removed by composing with another MSO-definable transformation. We can transform the SST-output structures which are edge-labeled strings over $O_T \subseteq \Gamma^*$ to a node-labeled string over Γ. This transformation is again FO-definable by taking a suitable number of copies of the input domain ($max\{|s| \mid s \in O_T\}$). Now Lemma 11 follows from the closure of FO-transformations under composition [10].

5 From FOT to aperiodic 1-bounded SST

The goal of this section is to prove the following lemma by showing a reduction from FO-definable transformations to aperiodic, 1-bounded SSTs. Due to space limitations, we only sketch the main ideas of the proof of this result.

▶ **Lemma 15.** *A string transformation is FO-definable* ***only if*** *it is definable by an aperiodic, 1-bounded SST.*

FO-types, heads and tails. The FO-K-type (K-type for short) of a string s is the set of FO sentences of quantifier rank at most K that are true in s. The set of K-types is finite (up to logical equivalence) [17]. We start with a key observation. Given an FO-transducer, an input string s and a position i in s, all the maximal paths of the output structure induced by nodes of the form j^c, for all copies c and input positions $j \leq i$ define substrings of the output of s. The starting (resp. ending) nodes of these substrings are respectively called i-head and i-tails. Consider the FO-transduction shown in Figure 2 till position 3. Suppose that we omit the positions and edges of the output graph post position 3. Upto position 3, the output graph consists of two strings: the first string is between the 3-head 1^1 and the 3-tail 3^1 and stores aa, while the second string is between the 3-head 3^2 and the 3-tail 2^3 and stores the string $abab$. The key observation of [5] is that any i-head j^c (resp. i-tail) is uniquely identified by the K-type τ_1 of the string $s[1{:}j)$, the label a of input position j, the copy c, and the K-type τ_2 of the string $s(j{:}i]$, for a bound K that depends only on the FO-transducer.

SST construction. The main lines of the SST construction of [5] is to use as many SST variables X_α as tuples $\alpha = (\tau_1, a, c, \tau_2)$. Since the sets of K-types, labels and copies are finite, then so is the set of variables. At each position i in the input s, the content of X_α computed by the SST is exactly the substring in between the i-head and i-tail identified by the tuple α. To define the variable update when incrementing position i, if the SST knows the K-types of the current prefix and suffix respectively, it can determine how the $(i+1)$-heads and $(i+1)$-tails are connected to the i-heads and i-tails, based on the FO-formulas that define the output edge relation. Let us now explain how the SST can compute the K-types of the prefix up to i and of the suffix from position i. It is known that the K-type of a string s_1s_2 only depends on the K-types of s_1 and s_2 respectively [17]. Therefore, to compute the K-type of the prefix up to $i+1$, the SST only needs to know the K-type of the prefix up to i and the input label at position $i+1$. Therefore, the states of the SST are K-types. To compute the K-type of the suffix, we equip our SST with look-aheads, defined by aperiodic finite state automata. We naturally extend the notion of aperiodic SST to aperiodic SST with look-aheads, and, as an intermediate result, show that removing look-aheads can be done while preserving aperiodicity (as well as 1-boundedness). From an FO-transducer, the construction therefore produces an SST T_{la} with look-ahead. The main difficulty is to show that T_{la} is aperiodic and 1-bounded.

Aperiodicity and 1-boundedness of T_{la}. One of the technical difficulties in showing that T_{la} is aperiodic is to show that it computes the type informations and update the variables in an aperiodic manner. A well-known property [17] we exploit is that for $m \geq 2^K$, for any string s, the strings s^m and s^{m+1} are indistinguishable by FO-sentences of rank at most K.

For the sake of understanding of this sketch and, in order to focus only on aperiodicity of variable updates, we rather assume, in this sketch, that the positions i of the input strings

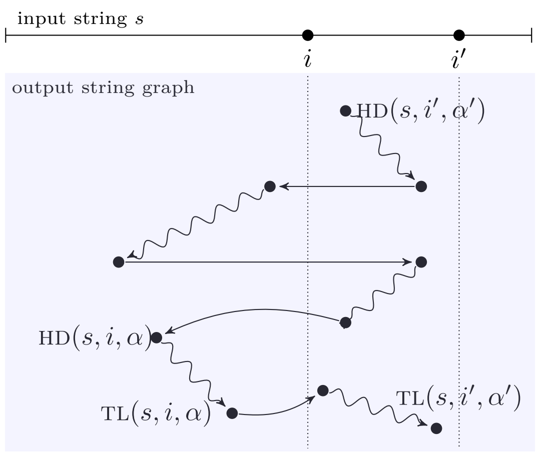

input string s
i
i'
output string graph
$\mathrm{HD}(s,i',\alpha')$
$\mathrm{HD}(s,i,\alpha)$
$\mathrm{TL}(s,i,\alpha)$
$\mathrm{TL}(s,i',\alpha')$

Figure 5 Variable flow is FO-definable.

s have been initially extended with type informations (τ_1, τ_2) where τ_1 is the K-type of $s[1{:}i]$ and τ_2 is the K-type of $s(i{:}|s|]$. Therefore, we can transform T_{la} into a one-state SST T (without look-ahead), assuming it gets as input only strings extended with valid type information. The 1-boundedness of T is a simple consequence of the construction (and was already shown in [5] through the notion of restricted copy). Let us now briefly explain why T is aperiodic, or equivalently, that its variable flow is aperiodic. It is sufficient to show that the variable flow is FO-definable. Given $s \in \Sigma^*$, a position i in s, and a tuple $\alpha = (\tau_1, a, c, \tau_2)$ as defined before, we denote by $\mathrm{HD}(s,i,\alpha)$ the i-head (resp. $\mathrm{TL}(s,i,\alpha)$ the i-tail) defined by α in s. Given another tuple α' and a position $i' > i$ in s, we relate the flow between variable X_α at position i to variable $X_{\alpha'}$ at position i' to the existence of a path from $\mathrm{HD}(s,i',\alpha')$ to $\mathrm{HD}(s,i,\alpha)$ that do not go beyond position i' in the output graph of s.

▶ **Example 16.** Consider the FO-transformation of Fig. 5. As a consequence of the invariant of our construction, the substring s_1 that starts in position $\mathrm{HD}(s,i,\alpha)$ and ends in $\mathrm{TL}(s,i,\alpha)$, at position i, is stored in variable X_α. The substring s_2 from $\mathrm{HD}(s,i',\alpha')$ to $\mathrm{TL}(s,i',\alpha')$ is stored, at position i', in variable $X_{\alpha'}$. Since s_1 is a substring of s_2, the content of variable X_α at position i (i.e. s_1) flows into the content of variable $X_{\alpha'}$ at position i' (i.e. s_2). Based on the fact that the output transition closure of the edge relation is defined in FO, and the fact that types are also FO-definable, we show that the existence of a path from $\mathrm{HD}(s,i',\alpha')$ to $\mathrm{HD}(s,i,\alpha)$ that do not cross position i' is FO-definable, and so is the variable flow.

As mentioned earlier, the complete proof starts directly with an SST with look-ahead that computes the type information, therefore one has to study both state flow and variable flow. An alternative proof could have been to compose two aperiodic SSTs (w/o lookaheads): the first one annotates the string with type information, and the second one is the one-state SST T. Then, it would remain to prove that aperiodic SSTs are closed under composition (which is a consequence of our result and the fact that FO-transducers are closed under composition). However, it is not clear that directly proving that aperiodic SSTs are closed under composition would have been simpler than our proof based on SSTs with look-aheads.

References

1 R. Alur and P. Černý. Expressiveness of streaming string transducers. In *Proc. FSTTCS 2010*, pages 1–12, 2010.

2 R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proc. POPL 2011*, pages 599–610, 2011.

3 R. Alur and L. D'Antoni. Streaming tree transducers. In *Proc. ICALP 2012*, pages 42–53, 2012.

4 R. Alur, L. D'Antoni, J. V. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *Proc. LICS 2013*, pages 13–22, 2013.

5 R. Alur, A. Durand-Gasselin, and A. Trivedi. From monadic second-order definable string transformations to transducers. In *Proc. LICS 2013*, pages 458–467, 2013.

6 R. Alur, E. Filiot, and A. Trivedi. Regular transformations of infinite strings. In *Proc. LICS 2012*, pages 65–74, 2012.

7 M. Bojanczyk. Transducers with origin information. In *Proc. ICALP 2014*, pages 26–37, 2014.

8 J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6(1–6):66–92, 1960.

9 O. Carton and L. Dartois. Aperiodic two-way transducers. In *Highlights of Logic, Automata and Games*, 2013. Slides available at `http://highlights-conference.org/pub/3-1-Dartois.pdf`.

10 B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

11 V. Diekert and P. Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, pages 261–306. Amsterdam University Press, 2008.

12 J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2:216–254, 2001.

13 J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing*, 32:950–1006, 2003.

14 E. Filiot, S. N. Krishna, and A. Trivedi. First-order definable string transformations.

15 P. McKenzie, T. Schwentick, D. Therien, and H. Vollmer. The many faces of a translation. *JCSS*, 72, 2006.

16 J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66:163–176, 1985.

17 H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

Regular Sensing

Shaull Almagor[1], Denis Kuperberg[2], and Orna Kupferman[1]

1 **The Hebrew University, Israel**
2 **The University of Warsaw, Poland**

Abstract

The size of deterministic automata required for recognizing regular and ω-regular languages is a well-studied measure for the complexity of languages. We introduce and study a new complexity measure, based on the *sensing* required for recognizing the language. Intuitively, the sensing cost quantifies the detail in which a random input word has to be read in order to decide its membership in the language. We show that for finite words, size and sensing are related, and minimal sensing is attained by minimal automata. Thus, a unique minimal-sensing deterministic automaton exists, and is based on the language's right-congruence relation. For infinite words, the minimal sensing may be attained only by an infinite sequence of automata. We show that the optimal limit cost of such sequences can be characterized by the language's right-congruence relation, which enables us to find the sensing cost of ω-regular languages in polynomial time.

1998 ACM Subject Classification F.4.3 Formal Languages, B.8.2 Performance Analysis and Design Aids, F.1.1 Models of Computation

Keywords and phrases Automata, regular languages, ω-regular languages, complexity, sensing, minimization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.161

1 Introduction

Studying the complexity of a formal language, there are several complexity measures to consider. When the language is given by means of a Turing Machine, the traditional measures are time and space demands. Theoretical interest as well as practical considerations have motivated additional measures, such as randomness (the number of random bits required for the execution) [9] or communication complexity (number and length of messages required) [8]. For regular and ω-regular languages, given by means of finite-state automata, the classical complexity measure is the size of a minimal deterministic automaton that recognizes the language.

We introduce and study a new complexity measure, namely the *sensing cost* of the language. Intuitively, the sensing cost of a language measures the detail with which a random input word needs to be read in order to decide membership in the language. Sensing has been studied in several other CS contexts. In theoretical CS, in methodologies such as PCP and property testing, we are allowed to sample or query only part of the input [6]. In more practical applications, mathematical tools in signal processing are used to reconstruct information based on compressed sensing [4], and in the context of data streaming, one cannot store in memory the entire input, and therefore has to approximate its properties according to partial "sketches" [10].

Our interest in regular sensing is motivated by the use of finite-state automata (as well as monitors, controllers, and transducers) in reasoning about on-going behaviors of reactive systems. In particular, a big challenge in the design of monitors is an optimization of the sensing needed for deciding the correctness of observed behaviors. Our goal is to formalize regular sensing in the finite-state setting and to study the sensing complexity measure for regular and ω-regular languages.

A natural setting in which sensing arises is *synthesis*: given a specification over sets I and O of input and output signals, the goal is to construct a finite-state system that, given a sequence of input signals, generates a computation that satisfies the specification. In each moment in time, the system reads an assignment to the input signals, namely a letter in 2^I, which requires the activation of $|I|$ Boolean sensors. A well-studied special case of limited sensing is synthesis with *incomplete information*. There, the system can read only a subset of the signals in I, and should still generate only computations that satisfy the specification [7, 2]. A more sophisticated case of sensing in the context of synthesis is studied in [3], where the system can read some of the input signals some of the time. In more detail, sensing the truth value of an input signal has a cost, the system has a budget for sensing, and it tries to realize the specification while minimizing the required sensing budget.

We study the fundamental questions on regular sensing. We consider languages over alphabets of the form 2^P, for a finite set P of signals. Consider a deterministic automaton $\mathcal{A}$ over an alphabet 2^P. For a state q of $\mathcal{A}$, we say that a signal $p \in P$ is *sensed* in q if at least one transition taken from q depends on the truth value of p. The *sensing cost* of q is the number of signals it senses, and the sensing cost of a run is the average sensing cost of states visited along the run. We extend the definition to automata by assuming a uniform distribution of the inputs.[1] Thus, the sensing cost of $\mathcal{A}$ is the limit of the expected sensing of runs over words of increasing length.[2] We show that this definition coincides with one that is based on the stationary distribution of the Markov chain induced by $\mathcal{A}$, which enables us to calculate the sensing cost of an automaton in polynomial time. The sensing cost of a language L, of either finite or infinite words, is then the infimum of the sensing costs of deterministic automata for L. In the case of infinite words, one can study different classes of automata, yet we show that the sensing cost is independent of the acceptance condition being used.

We start by studying the sensing cost of regular languages of finite words. For the complexity measure of size, the picture in the setting of finite words is very clean: each language L has a unique minimal deterministic automaton (DFA), namely the *residual automaton* $\mathcal{R}_L$ whose states correspond to the equivalence classes of the Myhill-Nerode right-congruence relation for L. We show that minimizing the state space of a DFA can only reduce its sensing cost. Hence, the clean picture of the size measure is carried over to the sensing measure: the sensing cost of a language L is attained in the DFA $\mathcal{R}_L$. In particular, since DFAs can be minimized in polynomial time, we can construct in polynomial time a minimally-sensing DFA, and can compute in polynomial time the sensing cost of languages given by DFAs.

We then study the sensing cost of ω-regular languages, given by means of deterministic parity automata (DPAs). Recall the size complexity measure. There, the picture for languages of infinite words is not clean: A language needs not have a unique minimal DPA, and the

[1] Our study and results apply also to a non-uniform distribution on the letters, given by a Markov chain (see Remark 19).

[2] Alternatively, one could define the sensing cost of $\mathcal{A}$ as the cost of its "most sensing" run. Such a worst-case approach is taken in [3], where the sensing cost needs to be kept under a certain budget in all computations, rather than in expectation. We find the average-case approach we follow appropriate for sensing, as the cost of operating sensors may well be amortized over different runs of the system, and requiring the budget to be kept under a threshold in every run may be too restrictive. Thus, the automaton must answer correctly for every word, but the sensing should be low only on average, and it is allowed to operate an expensive sensor now and then.

problem of finding one is NP-complete [12]. It turns out that the situation is challenging also in the sensing measure. First, we show that different minimal DPAs for a language may have different sensing costs. In fact, bigger DPAs may have smaller sensing costs.

Before describing our results, let us describe a motivating example that demonstrates the intricacy in the case of ω-regular languages. Consider a component in a vacuum-cleaning robot that monitors the dust collector and checks that it is empty infinitely often. The proposition *empty* indicates whether the collector is empty and a sensor needs to be activated in order to know its truth value. One implementation of the component would sense *empty* throughout the computation. This corresponds to the classical two-state DPA for "infinitely often *empty*". A different implementation can give up the sensing of *empty* for some fixed number k of states, then wait for *empty* to hold, and so forth. The bigger k is, the lazier is the sensing and the smaller the sensing cost is. As the example demonstrates, there may be a trade-off between the sensing cost of an implementation and its size. Other considerations, like a preference to have eventualities satisfied as soon as possible, enter the picture too.

Our main result is that despite the above intricacy, the sensing cost of an ω-regular language L is the sensing cost of the residual automaton $\mathcal{R}_L$ for L. It follows that the sensing cost of an ω-regular language can be computed in polynomial time. Unlike the case of finite words, it may not be possible to define L on top of $\mathcal{R}_L$. Interestingly, however, $\mathcal{R}_L$ does capture exactly the sensing required for recognizing L. The proof of this property of $\mathcal{R}_L$ is the main technical challenge of our contribution. The proof goes via a sequence $(\mathcal{B}_n)_{n=1}^{\infty}$ of DPAs whose sensing costs converge to that of L. The DPA $\mathcal{B}_n$ is obtained from a DPA $\mathcal{A}$ for L by a lazy sensing strategy that spends time in n copies of $\mathcal{R}_L$ between visits to $\mathcal{A}$, but spends enough time in $\mathcal{A}$ to ensure that the language is L. It is worth noting that this result is far from being intuitive. Indeed, first, as mentioned above, the extra expressive power that is added to the setting by the acceptance condition of DPAs makes the residual automaton irrelevant in the context of size minimization. Moreover, in the context of sensing, there need not be a single DPA that attains the minimal sensing cost. It is thus surprising that $\mathcal{R}_L$, which has no acceptance condition, captures the sensing cost of all DPAs. We believe that this reflects a general property of deterministic parity automata that could be useful outside of the scope of sensing. Intuitively, it means that we can "lose track" of the run of a deterministic automaton for arbitrary long periods, just keeping the residual in memory, and still be able to recognize the wanted language.

Due to lack of space, some of the proofs are omitted and can be found in the full version, in the authors' home pages.

2 Preliminaries

Automata

A *deterministic automaton on finite words* (DFA, for short) is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a total transition function, and $\alpha \subseteq Q$ is a set of accepting states. We sometimes refer to δ as a relation $\Delta \subseteq Q \times \Sigma \times Q$, with $\langle q, \sigma, q' \rangle \in \Delta$ iff $\delta(q, \sigma) = q'$. The run of $\mathcal{A}$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_m \in \Sigma^*$ is the sequence of states $q_0, q_1, \ldots, q_m$ such that $q_{i+1} = \delta(q_i, \sigma_{i+1})$ for all $i \geq 0$. The run is accepting if $q_m \in \alpha$. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if the run of $\mathcal{A}$ on w is accepting. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. For a state $q \in Q$, we use $\mathcal{A}^q$ to denote $\mathcal{A}$ with initial state q. We sometimes refer also to nondeterministic automata (NFAs), where $\delta : Q \times \Sigma \to 2^Q$ suggests several possible successor states. Thus, an NFA may have several runs on an input word w, and it accepts w if at least one of them is accepting.

Consider a language $L \subseteq \Sigma^*$. For two finite words u_1 and u_2, we say that u_1 and u_2 are *right L-indistinguishable*, denoted $u_1 \sim_L u_2$, if for every $z \in \Sigma^*$, we have that $u_1 \cdot z \in L$ iff $u_2 \cdot z \in L$. Thus, $\sim_L$ is the Myhill-Nerode right congruence used for minimizing automata. For $u \in \Sigma^*$, let $[u]$ denote the equivalence class of u in $\sim_L$ and let $\langle L \rangle$ denote the set of all equivalence classes. Each class $[u] \in \langle L \rangle$ is associated with the *residual language* $u^{-1}L = \{w : uw \in L\}$. When L is regular, the set $\langle L \rangle$ is finite, and induces the *residual automaton* of L, defined by $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$, with $\langle [u], a, [u \cdot a] \rangle \in \Delta_L$ for all $[u] \in \langle L \rangle$ and $a \in \Sigma$. Also, α contains all classes $[u]$ with $u \in L$. The DFA $\mathcal{R}_L$ is well defined and is the unique minimal DFA for L.

A *deterministic automaton on infinite words* is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q, q_0, and δ are as in DFA, and α is an acceptance condition. The run of $\mathcal{A}$ on an infinite input word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is defined as for automata on finite words, except that the sequence of visited states is now infinite. For a run $r = q_0, q_1, \ldots$, let $\mathit{inf}(r)$ denote the set of states that r visits infinitely often. Formally, $\mathit{inf}(r) = \{q : q = q_i \text{ for infinitely many } i\text{'s}\}$. We consider the following acceptance conditions. In a *Büchi* automaton, the acceptance condition is a set $\alpha \subseteq Q$ and a run r is accepting iff $\mathit{inf}(r) \cap \alpha \neq \emptyset$. Dually, in a *co-Büchi*, again $\alpha \subseteq Q$, but r is accepting iff $\mathit{inf}(r) \cap \alpha = \emptyset$. Finally, parity condition is a mapping $\alpha : Q \to [i, \ldots, j]$, for integers $i \leq j$, and a run r is accepting iff $\max_{q \in \mathit{inf}(r)} \{\alpha(q)\}$ is even.

We extend the right congruence $\sim_L$ as well as the definition of the residual automaton $\mathcal{R}_L$ to languages $L \subseteq \Sigma^\omega$. Here, however, $\mathcal{R}_L$ need not accept the language of L, and we ignore its acceptance condition.

Sensing

We study languages over an alphabet $\Sigma = 2^P$, for a finite set P of signals. A letter $\sigma \in \Sigma$ corresponds to a truth assignment to the signals. When we define languages over Σ, we use predicates on P in order to denote sets of letters. For example, if $P = \{a, b, c\}$, then the expression $(\texttt{True})^* \cdot a \cdot b \cdot (\texttt{True})^*$ describes all words over 2^P that contain a subword $\sigma_a \cdot \sigma_b$ with $\sigma_a \in \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$ and $\sigma_b \in \{\{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}\}$.

Consider an automaton $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$. For a state $q \in Q$ and a signal $p \in P$, we say that p is *sensed in* q if there exists a set $S \subseteq P$ such that $\delta(q, S \setminus \{p\}) \neq \delta(q, S \cup \{p\})$. Intuitively, a signal is sensed in q if knowing its value may affect the destination of at least one transition from q. We use $\mathit{sensed}(q)$ to denote the set of signals sensed in q. The *sensing cost* of a state $q \in Q$ is $\mathit{scost}(q) = |\mathit{sensed}(q)|$. [3]

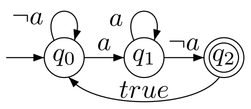
Consider a deterministic automaton $\mathcal{A}$ over $\Sigma = 2^P$ (and over finite or infinite words). For a finite run $r = q_1, \ldots, q_m$ of $\mathcal{A}$, we define the sensing cost of r, denoted $\mathit{scost}(r)$, as $\frac{1}{m} \sum_{i=1}^{m} \mathit{scost}(q_i)$. That is, $\mathit{scost}(r)$ is the average number of sensors that $\mathcal{A}$ uses during r. Now, for a finite word w, we define the sensing cost of w in $\mathcal{A}$, denoted $\mathit{scost}_{\mathcal{A}}(w)$, as the sensing cost of the run of $\mathcal{A}$ on w. Finally, the sensing cost of $\mathcal{A}$ is the expected sensing cost of words of length that tends to infinity, where we assume that the letters in Σ are uniformly distributed. Thus, $\mathit{scost}(\mathcal{A}) = \lim_{m \to \infty} |\Sigma|^{-m} \sum_{w : |w| = m} \mathit{scost}_{\mathcal{A}}(w)$. Note that the definition applies to automata on both finite and infinite words.

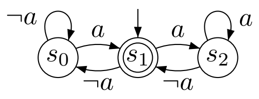
Two DFAs may recognize the same language and have different sensing costs. In fact, as we demonstrate in Example 1 below, in the case of infinite words two different minimal automata for the same language may have different sensing costs.

[3] We note that, alternatively, one could define the *sensing level* of states, with $\mathit{slevel}(q) = \frac{|\mathit{sensed}(q)|}{|P|}$. Then, for all states q, we have that $\mathit{slevel}(q) \in [0, 1]$. All our results hold also for this definition, simply by dividing the sensing cost by $|P|$.

For a language L of finite or infinite words, the sensing cost of L, denoted $scost(L)$ is the minimal sensing cost required for recognizing L by a deterministic automaton. Thus, $scost(L) = \inf_{\mathcal{A}:L(\mathcal{A})=L} scost(\mathcal{A})$. For the case of infinite words, we allow $\mathcal{A}$ to be a deterministic automaton of any type. In fact, as we shall see, unlike the case of succinctness, the sensing cost is independent of the acceptance condition used.

▶ **Example 1.** Let $P = \{a\}$. Consider the language $L \subseteq (2^{\{a\}})^\omega$ of all words with infinitely many a and infinitely many $\neg a$. In the following figure we present two minimal DBAs (deterministic Büchi automata) for L with different sensing costs.





While all the states of the second automaton sense a, thus its sensing cost is 1, the signal a is not sensed in all the states of the first automaton, thus its sensing cost is strictly smaller than 1 (to be precise, it is $\frac{4}{5}$, as we shall see in Example 7).

▶ **Remark 2.** Our study of sensing considers deterministic automata. The notion of sensing is less natural in the nondeterministic setting. From a conceptual point of view, we want to capture the number of sensors required for an actual implementation for recognizing the language. Technically, guesses can reduce the number of required sensors. To see this, take $P = \{a\}$ and consider the language $L = \texttt{True}^* \cdot a$. A DFA for L needs two states, both sensing a. An NFA for L can guess the position of the letter before the last one, where it moves to the only state that senses a. The sensing cost of such an NFA is 0 (for any reasonable extension of the definition of cost on NFAs). ◀

Probability

Consider a directed graph $G = \langle V, E\rangle$. A *strongly connected component* (SCC) of G is a maximal (with respect to containment) set $C \subseteq V$ such that for all $x, y \in C$, there is a path from x to y. An SCC (or state) is *ergodic* if no other SCC is reachable from it, and is *transient* otherwise.

An automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha\rangle$ induces a directed graph $G_\mathcal{A} = \langle Q, E\rangle$ in which $\langle q, q'\rangle \in E$ iff there is a letter σ such that $q' \in \delta(q, \sigma)$. When we talk about the SCCs of $\mathcal{A}$, we refer to those of $G_\mathcal{A}$. Recall that we assume that the letters in Σ are uniformly distributed, thus $\mathcal{A}$ also corresponds to a Markov chain $M_\mathcal{A}$ in which the probability of a transition from state q to state q' is $p_{q,q'} = \frac{1}{|\Sigma|}|\{\sigma \in \Sigma : \delta(q,\sigma) = q'\}|$. Let $\mathcal{C}$ be the set of $\mathcal{A}$'s SCC, and $\mathcal{C}_e \subseteq \mathcal{C}$ be the set of its ergodic SCC's.

Consider an ergodic SCC $C \in \mathcal{C}_e$. Let P_C be the matrix describing the probability of transitions in C. Thus, the rows and columns of P_C are associated with states, and the value in coordinate q, q' is $p_{q,q'}$. By [5], there is a unique probability vector $\pi_C \in [0,1]^C$ such that $\pi_C P_C = \pi_C$. This vector describes the *stationary distribution* of C: for all $q \in C$ it holds that $\pi_C(q) = \lim_{m\to\infty} \frac{E^C_m(q)}{m}$, where $E^C_m(q)$ is the average number of occurrences of q in a run of M_A of length m that starts anywhere in C [5]. Thus, intuitively, $\pi_C(q)$ is the probability that a long run that starts in C ends in q. In order to extend the distribution to the entire Markov chain of $\mathcal{A}$, we have to take into account the probability of reaching each of the ergodic components. The *SCC-reachability distribution* of $\mathcal{A}$ is the function $\rho : \mathcal{C} \to [0,1]$ that maps each ergodic SCC C of $\mathcal{A}$ to the probability that M_A eventually reaches C, starting from the initial state. We can now define the *limiting distribution* $\pi : Q \to [0,1]$, as

$$\pi(q) = \begin{cases} 0 & \text{if } q \text{ is transient,} \\ \pi_C(q)\rho(C) & \text{if } q \text{ is in some } C \in \mathcal{C}_e. \end{cases}$$

Note that $\sum_{q \in Q} \pi(q) = 1$, and that if P is the matrix describing the transitions of M_A and π is viewed as a vector in $[0,1]^Q$, then $\pi P = \pi$. Intuitively, the limiting distribution of state q describes the probability of a run on a random and long input word to end in q. Formally, we have the following lemma, whose proof appears in the full version.

▶ **Lemma 3.** *Let $E_m(q)$ be the expected number of occurrences of a state q in a run of length m of M_A that starts in q_0. Then, $\pi(q) = \lim_{m\to\infty} \frac{E_m(q)}{m}$.*

Computing The Sensing Cost of an Automaton

Consider a deterministic automaton $\mathcal{A} = \langle 2^P, Q, \delta, q_0, \alpha \rangle$. The definition of *scost*$(\mathcal{A})$ by means of the expected sensing cost of words of length that tends to infinity does not suggest an algorithm for computing it. In this section we show that the definition coincides with a definition that sums the costs of the states in $\mathcal{A}$, weighted according to the limiting distribution, and show that this implies a polynomial-time algorithm for computing *scost*$(\mathcal{A})$. This also shows that the cost is well-defined for all automata.

▶ **Theorem 4.** *For all automata $\mathcal{A}$, we have scost$(\mathcal{A}) = \sum_{q\in Q} \pi(q) \cdot$ scost(q), where π is the limiting distribution of $\mathcal{A}$.*

▶ **Remark 5.** *It is not hard to see that if $\mathcal{A}$ is strongly connected, then π is the unique stationary distribution of M_A and is independent of the initial state of $\mathcal{A}$. Accordingly, scost$(\mathcal{A})$ is also independent of $\mathcal{A}$'s initial state in this special case.* ◀

▶ **Theorem 6.** *Given an automaton $\mathcal{A}$, the sensing cost scost$(\mathcal{A})$ can be calculated in polynomial time.*

Proof. By Theorem 4, we have that *scost*$(\mathcal{A}) = \sum_{q\in Q} \pi(q) \cdot$ *scost*(q), where π is the limiting distribution of $\mathcal{A}$. By the definition of π, we have that $\pi(q) = \pi_C(q)\rho(C)$, if q is in some $C \in \mathcal{C}_e$. Otherwise, $\pi(q) = 0$. Hence, the computational bottleneck is the calculation of the SCC-reachability distribution $\rho : C \to [0,1]$ and the stationary distributions π_C for every $C \in \mathcal{C}_e$. It is well known that both can be computed in polynomial time via classic algorithms on matrices. For completeness, we give the details in the full version. ◀

▶ **Example 7.** Recall the first DBA described in Example 1. Its limiting distribution is $\pi(q_0) = \pi(q_1) = \frac{2}{5}$, $\pi(q_2) = \frac{1}{5}$. Accordingly, its cost is $1 \cdot \frac{2}{5} + 1 \cdot \frac{2}{5} + 0 \cdot \frac{1}{5} = \frac{4}{5}$.

Additional examples can be found in the full version.

3 The Sensing Cost of Regular Languages of Finite Words

In this section we study the setting of finite words. We show that there, sensing minimization goes with size minimization, which makes things clean and simple, as size minimization for DFAs is a feasible and well-studied problem. We also study theoretical properties of sensing. We show that, surprisingly, abstraction of signals may actually increase the sensing cost of a language, and we study the effect of classical operations on regular languages on their sensing cost. These last two contributions can be found in the full version.

Consider a regular language $L \subseteq \Sigma^*$, with $\Sigma = 2^P$. Recall that the residual automaton $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$ is the minimal-size DFA that recognizes L. We claim that $\mathcal{R}_L$ also minimizes the sensing cost of L.

▶ **Lemma 8.** *Consider a regular language $L \subseteq \Sigma^*$. For every DFA $\mathcal{A}$ with $L(\mathcal{A}) = L$, we have that $scost(\mathcal{A}) \geq scost(\mathcal{R}_L)$.*

Proof. Consider a word $u \in \Sigma^*$. After reading u, the DFA $\mathcal{R}_L$ reaches the state $[u]$ and the DFA $\mathcal{A}$ reaches a state q with $L(\mathcal{A}^q) = u^{-1}L$. Indeed, otherwise we can point to a word with prefix u that is accepted only in one of the DFAs. We claim that for every state $q \in Q$ such that $L(\mathcal{A}^q) = u^{-1}L$, it holds that $sensed([u]) \subseteq sensed(q)$. To see this, consider a signal $p \in sensed([u])$. By definition, there exists a set $S \subseteq P$ and words u_1 and u_2 such that $([u], S \setminus \{p\}, [u_1]) \in \Delta_L$, $([u], S \cup \{p\}, [u_2]) \in \Delta_L$, yet $[u_1] \neq [u_2]$. By the definition of $\mathcal{R}_L$, there exists $z \in (2^P)^*$ such that, w.l.o.g, $z \in u_1^{-1}L \setminus u_2^{-1}L$. Hence, as $L(\mathcal{A}^q) = u^{-1}L$, we have that $\mathcal{A}^q$ accepts $(S \setminus \{p\}) \cdot z$ and rejects $(S \cup \{p\}) \cdot z$. Let $\delta_{\mathcal{A}}$ be the transition function of $\mathcal{A}$. By the above, $\delta_{\mathcal{A}}(q, S \setminus \{p\}) \neq \delta_{\mathcal{A}}(q, S \cup \{p\})$. Therefore, $p \in sensed(q)$, and we are done. Now, $sensed([u]) \subseteq sensed(q)$ implies that $scost(q) \geq scost([u])$.

Consider a word $w_1 \cdots w_m \in \Sigma^*$. Let $r = r_0, \ldots, r_m$ and $[u_0], \ldots, [u_m]$ be the runs of $\mathcal{A}$ and $\mathcal{R}_L$ on w, respectively. Note that for all $i \geq 0$, we have $u_i = w_1 \cdot w_2 \cdots w_i$. For all $i \geq 0$, we have that $L(\mathcal{A}^{r_i}) = u_i^{-1}L$, implying that then $scost(r_i) \geq scost([u_i])$. Hence, $scost_{\mathcal{A}}(w) \geq scost_{\mathcal{R}_L}(w)$. Since this holds for every word in Σ^*, it follows that $scost(\mathcal{A}) \geq scost(\mathcal{R}_L)$. ◀

Since $L(\mathcal{R}_L) = L$, then $scost(L) \leq scost(\mathcal{R}_L)$. This, together with Lemma 8, enables us to conclude the following.

▶ **Theorem 9.** *For every regular language $L \subseteq \Sigma^*$, we have $scost(L) = scost(\mathcal{R}_L)$.*

Finally, since DFAs can be size-minimized in polynomial time, Theorems 6 and 9 imply we can efficiently minimize also the sensing cost of a DFA and calculate the sensing cost of its language:

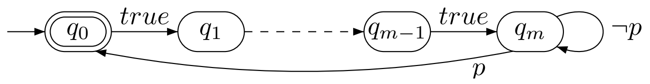
▶ **Theorem 10.** *Given a DFA $\mathcal{A}$, the problem of computing $scost(L(\mathcal{A}))$ can be solved in polynomial time.*

4 The Sensing Cost of ω-Regular Languages

For the case of finite words, we have a very clean picture: minimizing the state space of a DFA also minimizes its sensing cost. In this section we study the case of infinite words. There, the picture is much more complicated. In Example 1 we saw that different minimal DBAs may have a different sensing cost. We start this section by showing that even for languages that have a single minimal DBA, the sensing cost may not be attained by this minimal DBA, and in fact it may be attained only as a limit of a sequence of DBAs.

▶ **Example 11.** Let $P = \{p\}$, and consider the language L of all words $w_1 \cdot w_2 \cdots$ such that $w_i = \{p\}$ for infinitely many i's. Thus, $L = (\texttt{True}^* \cdot p)^\omega$. A minimal DBA for L has two states. The minimal sensing cost for a two-state DBA for L is $\frac{2}{3}$ (the classical two-state DBA for L senses p in both states and thus has sensing cost 1. By taking $\mathcal{A}_1$ in the sequence we shall soon define we can recognize L by a two-state DBA with sensing cost $\frac{2}{3}$). Consider the sequence of DBAs $\mathcal{A}_m$ appearing in the figure below. The DBA $\mathcal{A}_m$ recognizes $(\texttt{True}^{\geq m} \cdot p)^\omega$, which is equivalent to L, yet enables a "lazy" sensing of p. Formally, the stationary distribution π for $\mathcal{A}_m$ is such that $\pi(q_i) = \frac{1}{m+2}$ for $0 \leq i \leq m-1$ and

$\pi(q_m) = \frac{2}{m+2}$. In the states $q_0, \ldots, q_{m-1}$ the sensing cost is 0 and in q_m it is 1. Accordingly, $scost(\mathcal{A}_m) = \frac{2}{m+2}$, which tends to 0 as m tends to infinity.



4.1 Characterizing $scost(L)$ by the residual automaton for L

In this section we state and prove our main result, which characterizes the sensing cost of an ω-regular language by means of the residual automaton for the language:

▶ **Theorem 12.** *For every ω-regular language $L \subseteq \Sigma^\omega$, we have $scost(L) = scost(\mathcal{R}_L)$.*

The proof is described over the following section. The first direction, showing that $scost(L) \geq scost(\mathcal{R}_L)$, is proved by similar considerations to those used in the proof of Lemma 8 for the setting of finite words, and can be found in the full version.

Our main effort is to prove that $scost(L) \leq scost(\mathcal{R}_L)$. To show this, we construct, given a DPA $\mathcal{A}$ such that $L(\mathcal{A}) = L$, a sequence $(\mathcal{B}_n)_{n\geq 1}$ of DPAs such that $L(\mathcal{B}_n) = L$ for every $n \geq 1$, and $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$. We note that since the DPAs $\mathcal{B}_n$ have the same acceptance condition as $\mathcal{A}$, there is no trade-off between sensing cost and acceptance condition. More precisely, if L can be recognized by a DPA with parity ranks $[i, j]$ (in particular, if L is DBA-recognizable), then the sensing cost for $L(\mathcal{A})$ can be obtained by a DPA with parity ranks $[i, j]$.

We first assume that $\mathcal{A}$ is strongly connected. We will later show how to drop this assumption.

Let $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$ be a strongly connected DPA for L. We assume that $\mathcal{A}$ is minimally ranked. Thus, if $\mathcal{A}$ has parity ranks $\{0, 1, \ldots, k\}$, then there is no DPA for L with ranks $\{0, 1, \ldots, k-1\}$ or $\{1, 2, \ldots, k\}$. Also, if $\mathcal{A}$ has ranks $\{1, 2, \ldots, k\}$, we consider the complement DPA, which is $\mathcal{A}$ with ranks $\{0, 1, \ldots, k-1\}$. Since DPAs can be complemented by dualizing the acceptance condition, their sensing cost is preserved under complementation, so reasoning about the complemented DPA is sound. For $0 \leq i \leq k$, a cycle in $\mathcal{A}$ is called an *i-loop* if the maximal rank along the cycle is i. For $0 \leq i \leq j \leq k$, an *$[i, j]$-flower* is a state $q_{❀} \in Q$ such that for every $i \leq r \leq j$, there is an r-loop that goes through $q_{❀}$.

The following is an adaptation of a result from [11] to strongly connected DPAs:

▶ **Lemma 13.** *Consider a strongly-connected minimally-ranked DPA $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$ with ranks $\{0, \ldots, k\}$. Then, there is a DPA $\mathcal{D} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{D}} \rangle$ such that all the following hold.*

1. *For every state $s \in Q$, we have $L(\mathcal{A}^s) = L(\mathcal{D}^s)$. In particular, $\mathcal{A}$ and $\mathcal{D}$ are equivalent.*
2. *There exists $m \in \mathbb{N}$ such that $\mathcal{D}$ has ranks $\{0, ..., 2m + k\}$ and has a $[2m, 2m + k]$ flower.*

Proof. We start with the following claim, whose proof appears in the full version.

▶ **Claim 14.** *$\mathcal{A}$ does not have an equivalent DPA with ranks $\{1, \ldots, k+1\}$.*

Now, [11] proves the lemma for $\mathcal{A}$ that needs not be strongly connected and has no equivalent DPA with ranks $\{1, \ldots, k+1\}$. There, the DPA $\mathcal{D}$ has ranks in $\{0, ..., 2m + k + 1\}$, and has a $[2m, 2m + k]$-flower $q_{❀}$. We argue that since $\mathcal{A}$ is strongly connected, $\mathcal{D}$ has only ranks in $\{0, ..., 2m + k\}$.

By [11], if there exists $m \in \mathbb{N}$ and a DPA $\mathcal{D}$ that recognizes $L(\mathcal{A})$ and has a $[2m, 2m+k+1]$-flower, then $L(\mathcal{A})$ cannot be recognized by a DPA with ranks $\{1, ..., k + 2\}$. Observe that in this case, $L(\mathcal{A})$ cannot be recognized by a DPA with ranks $\{0,...,k\}$ as well, as by increasing the ranks by 2 we get a DPA with ranks $\{2,...,k+2\}$, contradicting the fact $L(\mathcal{A})$ cannot be recognized by a DPA with ranks in $\{1,...,k+2\}$. Hence, as $\mathcal{A}$ with ranks $\{0,...,k\}$ does exist, the DPA $\mathcal{D}$ cannot have a $[2m, 2m+k+1]$-flower.

Now, in our case, the DPA $\mathcal{A}$, and therefore also $\mathcal{D}$, is strongly-connected. Thus, if $\mathcal{D}$ has a state with rank $2m+k+1$, then the state $q_{❀}$ is in the same component with this state, and is therefore a $[2m, 2m+k+1]$ flower. By the above, however, $\mathcal{D}$ cannot have a $[2m, 2m+k+1]$ flower, implying that $\mathcal{D}$ has ranks in $\{0,...,2m+k\}$. ◀

Let $\mathcal{A}$ and $\mathcal{D}$ be as in Lemma 13, and $q_{❀}$ be the $[2m, 2m+k]$-flower in $\mathcal{D}$. Note that $\mathcal{A}$ and $\mathcal{D}$ have the same structure and differ only in their acceptance condition. Let $\Omega = \{0,...,2m+k\}$. For a word $w \in \Sigma^*$, let $\rho = s_1, s_2, ..., s_n$ be the run of $\mathcal{D}$ on w. If ρ ends in $q_{❀}$, we define the *$q_{❀}$-loop-abstraction of w* to be the rank-word $\text{abs}(w) \in \Omega^*$ of maximal ranks between successive visits to $q_{❀}$. Formally, let $w = y_0 \cdot y_1 \cdots y_t$ be a partition of w such that $\mathcal{D}$ visits the state $q_{❀}$ after reading the prefix $y_0 \cdots y_j$, for all $0 \leq j \leq t$, and does not visit $q_{❀}$ in other positions. Then, $\text{abs}(y_i)$, for $0 \leq i \leq t$, is the maximal rank read along y_i, and $\text{abs}(w) = \text{abs}(y_0) \cdot \text{abs}(y_1) \cdots \text{abs}(y_t)$. Recall that $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$, where $\langle L \rangle$ are the equivalence classes of the right-congruence relation on L, thus each state $[u] \in \langle L \rangle$ is associated with the language $u^{-1}L$ of words w such that $uw \in L$. We define a function $\varphi : Q \to \langle L \rangle$ that maps states of $\mathcal{A}$ to languages in $\langle L \rangle$ by $\varphi(q) = L(\mathcal{A}^q)$. Observe that φ is onto. We define a function $\gamma : \langle L \rangle \to Q$ that maps languages in $\langle L \rangle$ to states of $\mathcal{A}$ by arbitrarily choosing for every language $u^{-1}L \in \langle L \rangle$ a state in $\varphi^{-1}(u^{-1}L)$.

We define a sequence of words $u_{2m}, \ldots, u_{2m+k} \in \Omega^*$ as follows. The definition proceeds by an induction. Let $M = |Q|+1$. First, $u_{2m} = (2m)^M$. Then, for $2m < i \leq 2m+k$, we have $u_i = (i \cdot u_{i-1})^{M-1} \cdot i$. For example, if $m = 2$ and $|Q| = 2$, then $u_4 = 444$, $u_5 = 544454445$, $u_6 = 6544454445654445444456$, and so on. Let $\mathcal{P}$ be a DFA that accepts a (finite) word $w \in \Sigma^*$ iff the run of $\mathcal{D}$ on w ends in $q_{❀}$ and u_{2m+k} is a suffix of $abs(w)$, for the word $u_{2m+k} \in \Omega^*$ defined above. In the full version we describe how to construct $\mathcal{P}$, essentially by combining a DFA over that alphabet Ω that recognizes $\Omega^* \cdot u_{2m+k}$ with a DFA with state space $Q \times \Omega$ that records the highest rank visited between successive visits to $q_{❀}$ and thus abstracts words in Σ^*.

We can now turn to the construction of the DPAs $\mathcal{B}_n$. Recall that $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$, and let $\mathcal{P} = \langle \Sigma, Q_{\mathcal{P}}, t_0, \Delta_{\mathcal{P}}, \{t_{acc}\} \rangle$. For $n \geq 1$, we define $\mathcal{B}_n = \langle \Sigma, Q_n, \langle q_0, t_0 \rangle, \Delta_n, \alpha_n \rangle$ as follows. The states of $\mathcal{B}_n$ are $Q_n = (\langle L \rangle \times \{1, \ldots, n\}) \cup (Q \times (Q_{\mathcal{P}} \setminus \{t_{acc}\}))$, where t_{acc} is the unique accepting state of $\mathcal{P}$. We refer to the two components in the union as the *$\mathcal{R}_L$-component* and the *$\mathcal{D}$-component*, respectively. The transitions of $\mathcal{B}_n$ are defined as follows.

- Inside the $\mathcal{R}_L$-component: for every transition $\langle [u], a, [u'] \rangle \in \Delta_L$ and $i \in \{1, \ldots, n-1\}$, there is a transition $\langle ([u], i), a, ([u'], i+1) \rangle \in \Delta_n$.
- From the $\mathcal{R}_L$-component to the $\mathcal{D}$-component: for every transition $\langle [u], a, [u'] \rangle \in \Delta_L$, there is a transition $\langle ([u], n), a, (\gamma([u']), t_0) \rangle \in \Delta_n$.
- Inside the $\mathcal{D}$-component: for every transitions $\langle q, a, q' \rangle \in \Delta$ and $\langle t, a, t' \rangle \in \Delta_{\mathcal{P}}$ with $t' \neq t_{acc}$, there is a transition $\langle (q, t), a, (q', t') \rangle \in \Delta_n$.
- From the $\mathcal{D}$-component to the $\mathcal{R}_L$-component: for every transitions $\langle q, a, q' \rangle \in \Delta$ and $\langle t, a, t_{acc} \rangle \in \Delta_{\mathcal{P}}$, there is a transition $\langle (q, t), a, (\varphi(q'), 1) \rangle \in \Delta_n$.

The acceptance condition of $\mathcal{B}_n$ is induced by that of $\mathcal{A}$. Formally $\alpha_n(q, t) = \alpha_{\mathcal{A}}(q)$, for states $(q, t) \in Q \times Q_{\mathcal{P}}$, and $\alpha_n([u], i) = 0$ for states $([u], i) \in \langle L \rangle \times \{1, \ldots, n\}$.

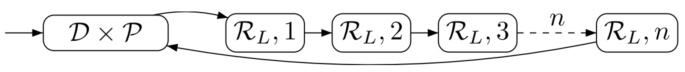


Figure 1 The DPA $\mathcal{B}_n$.

The idea behind the construction of $\mathcal{B}_n$ is as follows. The automaton $\mathcal{B}_n$ stays in $\mathcal{R}_L$ for n steps, then proceeds to a state in $\mathcal{D}$ with the correct residual language, and simulates $\mathcal{D}$ until the ranks corresponding to the word u_{2m+k} have been seen. It then goes back to $\mathcal{R}_L$, by projecting the current state of $\mathcal{D}$ onto its residual in $\langle L \rangle$. The bigger n is, the more time a run spends in the $\mathcal{R}_L$-component, making $\mathcal{R}_L$ the more dominant factor in the sensing cost of $\mathcal{B}_n$. As n tends to infinity, the sensing cost of $\mathcal{B}_n$ tends to that of $\mathcal{R}_L$. The technical challenge is to define $\mathcal{P}$ in such a way that even though the run spends less time in the $\mathcal{D}$ component, we can count on the ranks visited during this short time in order to determine whether the run is accepting. We are now going to formalize this intuition, and we start with the most challenging part of the proof, namely the equivalence of $\mathcal{B}_n$ and $\mathcal{A}$. The proof is decomposed into the three Lemmas 15, 16, and 17.

▶ **Lemma 15.** *Consider a word $u \in \Sigma^*$ such that the run of $\mathcal{B}_n$ on u reaches the $\mathcal{D}$-component in state $\langle q, t \rangle$. Then, $L(\mathcal{D}^q) = L(\mathcal{A}^q) = u^{-1}L$.*

Proof. We prove a stronger claim, namely that if the run of $\mathcal{B}_n$ on u ends in the $\mathcal{R}_L$-component in a state $\langle s, i \rangle$, then $s = [u]$, and if the run ends in the $\mathcal{D}$-component in a state $\langle q, t \rangle$, then $L(\mathcal{A}^q) = u^{-1}L$. The proof proceeds by induction on $|u|$ and is detailed in the full version. By Lemma 13, for every $q \in Q$, we have $L(\mathcal{A}^q) = L(\mathcal{D}^q)$, so the claim follows. ◀

▶ **Lemma 16.** *If the run of $\mathcal{B}_n$ on a word $w \in \Sigma^\omega$ visits the $\mathcal{R}_L$-component finitely many times, then $w \in L$ iff $w \in L(\mathcal{B}_n)$.*

Proof. Let $u \in \Sigma^*$ be a prefix of w such that the run of $\mathcal{B}_n$ on w stays forever in the $\mathcal{D}$-component after reading u. Let $(q, t) \in Q_n$ be the state reached by $\mathcal{B}_n$ after reading u. By Lemma 15, we have $L(\mathcal{A}^q) = u^{-1}L$. Since the run of $\mathcal{B}_n$ from (q, t) stays in the $\mathcal{D}$-components where it simulates the run of $\mathcal{A}$ from q, then $\mathcal{A}^q$ accepts the suffix $w^{|u|}$ iff $\mathcal{B}_n^{(q,t)}$ accepts $w^{|u|}$. It follows that $w \in L$ iff $w \in L(\mathcal{B}_n)$. ◀

The complicated case is when the run of $\mathcal{B}_n$ on w does visit the $\mathcal{R}_L$-component infinitely many times. This is where the special structure of $\mathcal{P}$ guarantees that the sparse visits in the $\mathcal{D}$-component are sufficient for determining acceptance.

▶ **Lemma 17.** *If the run of $\mathcal{B}_n$ on a word $w \in \Sigma^\omega$ visits the $\mathcal{R}_L$-component infinitely many times, then $w \in L$ iff $w \in L(\mathcal{B}_n)$.*

Proof. Let $\tau = s_1, s_2, s_3, \ldots$ be the run of $\mathcal{B}_n$ on w and let $\rho = q_1, q_2, q_3 \ldots$ be the run of $\mathcal{A}$ on w. We denote by $\tau[i, j]$ the infix $s_i, ..., s_j$ of τ. We also extend $\alpha_\mathcal{D}$ to (infixes of) runs by defining $\alpha_\mathcal{D}(\tau[i, j]) = \alpha_\mathcal{D}(s_i), ..., \alpha_\mathcal{D}(s_j)$. For a rank-word $u \in \Omega^*$, we say that an infix $\tau[i, j]$ is a *u-infix* if $\alpha_\mathcal{D}(\tau[i, j]) = u$.

If $v = \tau[i, j]$, for some $0 \leq i \leq j$, is a part of a run of $\mathcal{D}$ that consists of loops around $q_{❀}$, we define the *loop type of v* to be the word in Ω^* that describes the highest rank of each simple loop around $q_{❀}$ in v. An infix of τ whose loop type is u_i for some $2m \leq i \leq 2m + k$ is called a *u_i-loop-infix*.

By our assumption, τ contains infinitely many u_{2m+k}-infixes. Indeed, by the definition of $\mathcal{P}$, otherwise τ get trapped in the $\mathcal{D}$-component. We proceed by establishing a connection between u_i-loop-infixes of τ and the corresponding infixes of ρ, for all $2m \leq i \leq 2m + k$.

Let $i \in \{2m, \dots, 2m+k\}$, and consider a u_i-loop-infix. By the definition of u_i, such a u_i-loop-infix consists of a sequence of $M = |Q|+1$ i-loops in τ, with loops of lower ranks between them. We can write $w = xvw'$, where $v = w[c,d]$ is the sub word that corresponds to the u_i-loop-infix. Let $u_i' = \alpha_{\mathcal{A}}(\rho[c,d])$ be the ranks of ρ in its part that corresponds to v.

By our choice of M, we can find two indices $c \leq j < l \leq d$ such that the pairs $\langle (q_j, t), q_j' \rangle$ and $\langle (q_l, t'), q_l' \rangle$ reached by (τ, ρ) in indices j and l, respectively, satisfy $q_j = q_l = q_{❀}$ and $q_j' = q_l'$. Additionally, being a part of the run on a u_i-loop-infix, the highest rank seen between q_j and q_l in τ is i. We write $v = v_1 v_2 v_3$, where $v_1 = v[1,j]$, $v_2 = v[j+1,l]$, and $v_3 = v[l+1,|v|]$. Thus, the loop type of v_2 is in $(iu_{i-1})^+ i$, with the convention $u_{2m-1} = \epsilon$.

Consider the runs μ and η of $\mathcal{D}^{q_j}$ and of $\mathcal{A}^{q_j'}$ on v_2^ω, respectively. These runs are loops labeled by v_2, where the highest rank in μ is i. By Lemma 15, $L(\mathcal{D}^{q_j}) = L(\mathcal{D}^{q_j'}) = L(\mathcal{A}^{q_j'})$, so the highest rank in η must have same parity (odd or even) as i.

Thus, we showed that for every $i \in \{2m, ..., 2m+k\}$, and for every u_i-loop-infix v of τ, there is an infix of v with loop-type in $(iu_{i-1})^+ i$, such that the infix of ρ corresponding to v has highest rank of same parity as i.

We want to show that rank k is witnessed on ρ during every u_{2m+k}-infix of τ. Assume by way of contradiction that this is not the case. This means that there is some u_{2m+k}-infix v' in τ such that all ranks visited in ρ along v' are at most $k-2$. Indeed, since the highest rank has to be of the same parity as $2m+k$, which has the same parity as k, it cannot be $k-1$. By the same argument, within v' there is an infix v'' of u_{2m+k-1} of the form $((2m+k-1)(u_{2m+k-2}))^+(2m+k-1)$ in which the highest rank in ρ is of the same parity as $k-1$. As v'' is also an infix of v', the highest rank in ρ along v'' is at most $k-2$. Thus, the highest rank along v'' is at most $k-3$. By continuing this argument by induction down to 0, we reach a contradiction (in fact it is reached at level 1), as no rank below 0 is available.

We conclude that the run ρ witnesses a rank k in any u_k-infix of τ. Since τ contains infinitely many u_k-infixes, then ρ contains infinitely many ranks k, and, depending on the parity of k, either both ρ and τ are rejecting or both are accepting.

This concludes the proof that $w \in L$ iff $w \in L(\mathcal{B}_n)$. ◀

We proceed to show that the sensing cost of the sequence of DPAs $\mathcal{B}_n$ indeed converges to that of $\mathcal{R}_L$.

▶ **Lemma 18.** $\lim_{n\to\infty} \mathit{scost}(\mathcal{B}_n) = \mathit{scost}(\mathcal{R}_L)$.

Proof. Since $\mathcal{D}$ is strongly connected, then $q_{❀}$ is reachable from every state in $\mathcal{D}$. Also, since $q_{❀}$ is a $[2m, 2m+k]$-flower, we can construct a sequence of loops around $q_{❀}$ whose ranks correspond to the word u_{2m+k}. Thus, t_{acc} is reachable from every state in the $\mathcal{D}$-component. This implies that $\mathcal{B}_n$ is strongly connected, and therefore, a run of $\mathcal{B}_n$ is expected to traverse both components infinitely often, making the $\mathcal{R}_L$-component more dominant as n grows, implying that $\lim_{n\to\infty} \mathit{scost}(\mathcal{B}_n) = \mathit{scost}(\mathcal{R}_L)$. Formalizing this intuition involves a careful analysis of $\mathcal{B}_n$'s Markov chain, as detailed in the full version. ◀

Lemmas 16, and 17 put together ensure that for strongly connected automata, we have that $L(\mathcal{B}_n) = L$, so with Lemma 18, we get $\mathit{scost}(L) = \mathit{scost}(\mathcal{R}_L)$.

It is left to remove the assumption about $\mathcal{A}$ being strongly connected. The proof is detailed in the full version, and uses the above result on each ergodic component of $\mathcal{A}$.

▶ **Remark 19.** *All our results can be easily extended to a setting with a non-uniform distribution on the letters given by any Markov chain, or with a different cost for each input in each state. We can also use a decision tree to read the inputs instead of reading them* *simulatenously, defining for instance a cost of* 1.5 *if the state starts by reading* a*, then if* a *is true it also reads* b*.* ◀

5 Directions for Future Research

Regular sensing is a basic notion, which we introduced and studied for languages of finite and infinite words. In this section we discuss possible extensions and variants of our definition and contribution.

Open systems: Our setting assumes that all the signals in P are generated by the environment and read by the automaton. In the setting of open systems, we partition P into a set I of input signals, generated by the environment, and a set O of output signals, generated by the system. Then, we define the sensing cost of a specification as the minimal sensing cost required for a transducer that realizes it, where here, sensing is measured only with respect to the signals in I. Also, the transducer does not have to generate all the words in the language – it only has to associate a computation in the language with each input sequence. These two differences may lead to significantly different results than those presented in the paper.

Trade-off between sensing and quality: The key idea in the proof of Theorem 12 is that when we reason about languages of infinite words, it is sometimes possible to delay the sensing and only sense in "sparse" intervals. In practice, however, it is often desirable to satisfy eventualities quickly. This is formalized in multi-valued formalisms such as LTL with future discounting [1], where formulas assign higher satisfaction values to computations that satisfy eventualities fast. Our study here suggests that lower sensing leads to lower satisfaction values. An interesting problem is to study and formalize this intuitive trade-off between sensing and quality.

Transient cost: In our definition of sensing, transient states are of no importance. Consequently, for example, all safety languages have sensing cost 0, as the probability of a safety property not being violated is 0, and once it is violated, no sensing is required. An alternative definition of sensing cost may take transient states into an account. One way to do it is to define the sensing cost of a run as the discounted sum $\sum_{i\geq 0} 2^{-i} \cdot \mathit{sensed}(|q_i|)$ of the sensing costs of the states $q_0, q_1, \ldots$ it visits.

Beyond regular: Our definition of sensing cost can be adapted to more complex models, such as pushdown automata or Turing machines. It would be interesting to see the trade-off between sensing and classical complexity measures in such models.

References

1 S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. In *TACAS*, Lecture Notes in Computer Science. Springer, 2014.

2 K. Chatterjee and R. Majumdar. Minimum attention controller synthesis for omega-regular objectives. In *FORMATS*, pages 145–159, 2011.

3 K. Chatterjee, R. Majumdar, and T. A. Henzinger. Controller synthesis with budget constraints. In *HSCC*, volume 4981 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2008.

4 D. L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52:1289–1306, 2006.

5 C. Grinstead and J. Laurie Snell. *Introduction to Probability*, chapter 11 (Markov Chains), pages 405–470. American Mathematical Society, 1997.

6 G. Kindler. *Property Testing, PCP, and Juntas.* PhD thesis, Tel Aviv University, 2002.

7 O. Kupferman and M. Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, 1999.

8 E. Kushilevitz and N. Nisan. *Communication complexity.* Cambridge University Press, 1997.

9 C. Mauduit and A. Sárköz. On finite pseudorandom binary sequences. i. measure of pseudorandomness, the legendre symbol. *Acta Arith.*, 82(4):365–377, 1997.

10 S. Muthukrishnan. Theory of data stream computing: where to go. In *Proc. 30th Symposium on Principles of Database Systems*, pages 317–319, 2011.

11 D. Niwinski and I. Walukiewicz. Relating hierarchies of word and tree automata. In *STACS*, volume 1373 of *Lecture Notes in Computer Science*. Springer, 1998.

12 S. Schewe. Beyond Hyper-Minimisation – Minimising DBAs and DPAs is NP-Complete. In *FSTTCS*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, 2010.

Symbolic Solving of Extended Regular Expression Inequalities

Matthias Keil and Peter Thiemann

Institute for Computer Science
University of Freiburg
Freiburg, Germany
{keilr,thiemann}@informatik.uni-freiburg.de

Abstract

This paper presents a new algorithm for the containment problem for *extended regular expressions* that contain intersection and complement operators and that range over infinite alphabets. The algorithm solves extended regular expressions inequalities symbolically by term rewriting and thus avoids the translation to an expression-equivalent automaton.

Our algorithm is based on Brzozowski's regular expression derivatives and on Antimirov's term-rewriting approach to check containment. To deal with large or infinite alphabets effectively, we generalize Brzozowski's derivative operator to work with respect to (potentially infinite) representable character sets.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Extended regular expression, containment, infinite alphabet, infinite character set, effective boolean algebra

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.175

1 Introduction

Regular expressions have many applications in the context of software development and information technology: text processing, program analysis, compiler construction, query processing, and so on. Modern programming languages either come with standard libraries for regular expression processing or they provide built-in facilities to this end (e.g., Perl, Ruby, and JavaScript). Many of these implementations augment the basic regular operations $+$, $\cdot$, and * (union, concatenation, and Kleene star) with enhancements like character classes and wildcard literals, cardinalities, sub-matching, intersection, complement, and so on.

Regular expressions (RE) are advantageous in these domains because they provide a concise means to encode many interesting problems. REs are well suited for verification applications, because there are decision procedures for many problems involving them: the word problem ($w \in [\![r]\!]$), emptiness ($[\![r]\!] = \emptyset$), finiteness, containment ($[\![r]\!] \subseteq [\![s]\!]$), and equivalence ($[\![r]\!] = [\![s]\!]$). Here we let r and s range over RE and write $[\![\cdot]\!]$ for the function that maps a regular expression to the regular language that it denotes. There are also effective constructions for operations like union, intersection, complement, prefixes, suffixes, etc on regular languages.

Recent applications impose new demands on operations involving regular expressions. The Unicode character set with its more than 1.1 million code points requires the ability to deal effectively with very large character sets and hence character classes. Similarly, formalizing access contracts for objects in scripting languages even requires regular expressions over an infinite alphabet: in this application, the alphabet itself is an infinite formal language (the language of field names) and a “character class” (i.e., a set of field names) is itself described by a regular expression [12, 8]. Hence, a “character class” may have infinitely many elements.

To enable such applications, we study the containment problem for regular expressions with two enhancements. First, we consider *extended regular expressions* (ERE) that contain intersection and complement operators beyond the standard regular operators of union, concatenation, and Kleene star. An ERE also denotes a regular language but it can be much more concise than a standard RE. Second, we consider EREs on any alphabet that is presented as an effective boolean algebra. This extension encompasses some practically useful instances of infinite alphabets like the set of all field names in a scripting language.

The first enhancement is known to be decidable, but we give a new symbolic decision procedure based on Brzozowski’s regular expression derivatives [4] and Antimirov’s rewriting approach to check containment [1]. The second enhancement has been studied previously [21, 19, 20], but in the context of automata and finite state transducers. It has not been investigated at the level of regular expressions and in particular not in the context of Brzozowski’s and Antimirov’s work. We give sufficient conditions to ensure applicability of our modification of Brzozowski’s and Antimirov’s approach to the containment problem while retaining decidability.

1.1 Related Work

The practical motivation for considering this extension is drawn from the authors’ previous work on checking access contracts for objects in a scripting language at run time [12]. In that work, an access contract specifies a set of access paths that start from a specific anchor object. An access path is a word over the field names of the objects traversed by the path and we specify such a set of paths by a regular expression on the field names. We claim that such a regular expression draws from an infinite alphabet because a field name in a scripting language is an arbitrary string (of characters). For succinctness, we specify sets of field names using a second level of regular expressions on characters.

In our implementation, checking containment is required to reduce memory consumption. If the same object is restricted by more than one contract, then we apply containment checking to remove redundant contracts. In our previous work, contracts were limited to basic regular expressions and the field-level expressions were limited to disjunctions of literals. Applying the results of the present paper enables us to lift both restrictions.

The textbook approach to checking regular expression containment is via translation to finite automata, which may involve an exponential blowup, and then by constructing a simulation (or a bisimulation for equivalence) [9]. A related approach based on non-deterministic automata is presented by Bonchi and Pous [3].

The exponential blowup is due to the construction of a deterministic automaton from the regular expression. Thompson’s construction [18], creates a non-deterministic finite automaton with ϵ-transitions where the number of states and transitions is linear to the length of the (standard) regular expression. Glushkov’s [7] and McNaughton and Yamada’s [14] position automaton computes an $n+1$-state non-deterministic automaton with up to n^2 transitions from an n-symbol expression. They are the first to use the notion of a first symbol. Brzozowski’s regular expression derivatives [4] directly calculate a deterministic automaton from an ERE. Antimirov’s partial derivative approach [2] computes a $n+1$-state non-deterministic automation, but his work does not consider intersection and complement. We are not aware of an extension of Glushkov’s algorithm to extended regular expressions.

Owens and other have implemented an extension of Brzozowski’s approach with character classes and wildcards [16].

Antimirov [1] also proposes a symbolic method for solving regular expression inequalities, based on partial derivatives, with exponential worst-case run time. His *containment calculus* is closely related to the simulation technique used by Hopcroft and Karp [9] for proving equivalence of automata. In fact, a decision procedure for containment of regular expressions leads to one for equivalence and vice versa. Ginzburg [6] gives an equivalence procedure based on Brzozowski derivatives. Antimirov's original work does not consider intersection and complement. Caron and coworkers [5] extend Antimirov's work to ERE using antichains, but the resulting procedure is very complex compared to ours.

A shortcoming of all existing approaches is their restriction to finite alphabets. Supporting both makes a significant difference in practice: an iteration over the alphabet Σ is feasible for small alphabets, but it is impractical for very large alphabets (e.g., Unicode) or infinite ones (e.g., another level of regular languages as for our contracts). Furthermore, most regular expressions used in practice contain character sets. We apply techniques developed for symbolic finite automata to address these issues [20].

1.2 Overview

This paper is organized as follows. In Section 2, we recall notations and concepts. Section 3 introduces the notion of an effective boolean algebra for representing sets of symbols abstractly. Section 4 explains Antimirov's algorithm for checking containment, which is the starting point of our work. Next, Section 5 defines two notions of derivatives on regular expressions with respect to symbol sets. It continues to introduce the key notion of *next literals*, which ensures finiteness of our extension to Antimirov's algorithm. Section 6 contains the heart of our extended algorithm, a deduction system that determines containment of extended regular expressions along with a soundness proof.

A technical report [13] extends this paper by an appendix with further technical details, examples, and proofs of theorems.

2 Regular Expressions

An *alphabet* Σ is a denumerable, potentially infinite set of symbols. Σ^* is the set of all finite words over symbols from Σ with ϵ denoting the empty word. Let $a, b, c \in \Sigma$ range over symbols; $u, v, w \in \Sigma^*$ over words; and $A, B, C \subseteq \Sigma$ over sets of symbols.

Let $\mathcal{L}, \mathcal{L}' \subseteq \Sigma^*$ be languages. The *left quotient* of $\mathcal{L}$ by a word u, written $u^{-1}\mathcal{L}$, is the language $\{v \mid uv \in \mathcal{L}\}$. It is immediate from the definition that $(au)^{-1}\mathcal{L} = u^{-1}(a^{-1}\mathcal{L})$ and that $u \in \mathcal{L}$ iff $\epsilon \in u^{-1}\mathcal{L}$. Furthermore, $\mathcal{L} \subseteq \mathcal{L}'$ iff $u^{-1}\mathcal{L} \subseteq u^{-1}\mathcal{L}'$ for all words $u \in \Sigma^*$. The left quotient of one language by another is defined by $\mathcal{L}^{-1}\mathcal{L}' = \{v \mid uv \in \mathcal{L}', u \in \mathcal{L}\}$. We write $\mathcal{L}{\cdot}\mathcal{L}'$ for the concatenation of languages $\{uv \mid u \in \mathcal{L}, v \in \mathcal{L}'\}$ and $\mathcal{L}^*$ for the Kleene closure $\{v_1 \dots v_n \mid n \in \mathbb{N}, v_i \in \mathcal{L}\}$. We sometimes write $\overline{\mathcal{L}}$ for the complement $\Sigma^* \setminus \mathcal{L}$ and $\overline{A}$ for $\Sigma \setminus A$.

An *extended regular expression* (ERE) on an alphabet Σ is a syntactic phrase derivable from non-terminals r, s, t. It comprises the the empty word, literals, union, concatenation, Kleene star, as well as intersection and negation operators.

$$r, s, t \;:=\; \epsilon \mid A \mid r{+}s \mid r{\cdot}s \mid r^* \mid r\&s \mid !r$$

Compared to standard definitions, a *literal* is a set A of symbols, which stands for an abstract, possibly empty, character class. We write a instead of $\{a\}$ for the frequent case of a single letter literal. We consider regular expressions up to similarity [4], that is, up to associativity and commutativity of the union operator with the empty set as identity.

The language $[\![r]\!] \subseteq \Sigma^*$ of a regular expression r is defined inductively by:

$$\begin{array}{lll} [\![\epsilon]\!] = \{\epsilon\} & [\![r{+}s]\!] = [\![r]\!] \cup [\![s]\!] & [\![r\&s]\!] = [\![r]\!] \cap [\![s]\!] \\ [\![A]\!] = \{a \mid a \in A\} & [\![r{\cdot}s]\!] = [\![r]\!]{\cdot}[\![s]\!] & [\![!r]\!] = \overline{[\![r]\!]} \\ & [\![r^*]\!] = [\![r]\!]^* & \end{array}$$

For finite alphabets, $[\![r]\!]$ is a regular language. For arbitrary alphabets, we *define* a language to be regular, if it is equal to $[\![r]\!]$, for some ERE r.

We write $r \sqsubseteq s$ (r is *contained* in s) to express that $[\![r]\!] \subseteq [\![s]\!]$.

The *nullable* predicate $\nu(r)$ indicates whether $[\![r]\!]$ contains the empty word, that is, $\nu(r)$ iff $\epsilon \in [\![r]\!]$. It is defined inductively by:

$$\begin{array}{lllllllll} \nu(\epsilon) & = & \mathit{true} & \nu(r{+}s) & = & \nu(r) \vee \nu(s) & \nu(r\&s) & = & \nu(r) \wedge \nu(s) \\ \nu(A) & = & \mathit{false} & \nu(r{\cdot}s) & = & \nu(r) \wedge \nu(s) & \nu(!r) & = & \neg\nu(r) \\ & & & \nu(r^*) & = & \mathit{true} & & & \end{array}$$

The *Brzozowski derivative* $\partial_a(r)$ of an ERE r w.r.t. a symbol a computes a regular expression for the left quotient $a^{-1}[\![r]\!]$ (see [4]). It is defined inductively as follows:

$$\begin{array}{ll} \partial_a(\epsilon) = \emptyset & \partial_a(r{\cdot}s) = \begin{cases} \partial_a(r){\cdot}s{+}\partial_a(s), & \nu(r) \\ \partial_a(r){\cdot}s, & \neg\nu(r) \end{cases} \\ \partial_a(A) = \begin{cases} \epsilon, & a \in A \\ \emptyset, & a \notin A \end{cases} & \partial_a(r^*) = \partial_a(r){\cdot}r^* \\ & \partial_a(r\&s) = \partial_a(r)\&\partial_a(s) \\ \partial_a(r{+}s) = \partial_a(r){+}\partial_a(s) & \partial_a(!r) = !\partial_a(r) \end{array}$$

The case for the set literal A generalizes Brzozowski's definition. The definition is extended to words by $\partial_{au}(r) = \partial_u(\partial_a(r))$ and $\partial_\epsilon(r) = r$. It is easy to see that $u \in [\![r]\!]$ iff $\epsilon \in [\![\partial_u(r)]\!]$.

3 Representing Sets of Symbols

The definition of an ERE in Section 2 just states that a literal is a set of symbols $A \subseteq \Sigma$. However, to define tractable algorithms, we require that A is an element of an effective boolean algebra [20] $(U, \sqcup, \sqcap, \bar{\cdot}, \bot, \top)$ where $U \subseteq \wp(\Sigma)$ is closed under the boolean operations. Here $\sqcup$ and $\sqcap$ denote union and intersection of symbol sets, $\bar{\cdot}$ the complement, and $\bot$ and $\top$ the empty set and the full set Σ, respectively. In this algebra, we need to be able to decide equality of sets (hence the term *effective*) and to represent singleton symbols.

- For finite (small) alphabets, we may just take $U = \wp(\Sigma)$. A set of symbols may be enumerated and ranges of symbols may be represented by character classes, as customarily supported in regular expression implementations. Alternatively, a bitvector representation may be used.
- If the alphabet is infinite (or just too large), then the boolean algebra of finite and cofinite sets of symbols is the basis for a suitable representation. That is, the set $U = \{A \in \wp(\Sigma) \mid A \text{ finite} \vee \overline{A} \text{ finite}\}$ is effectively closed under the boolean operations.
- In our application to checking access contracts in scripting languages [12], the alphabet itself is a set of words (the field names of objects) composed from another set Γ of symbols: $\Sigma \subseteq \wp(\Gamma^*)$. To obtain an effective boolean algebra, we choose the set $U = \{A \subseteq \wp(\Gamma^*) \mid A \text{ is regular}\}$, which is effectively closed under the boolean operations.
- Sets of symbols may also be represented by formulas drawn from a decidable first-order theory over a (finite or infinite) alphabet. For example, the character range `[a-z]` would be represented by the formula $x \geq \texttt{'a'} \wedge x \leq \texttt{'z'}$. In this case, the boolean operations get

mapped to the disjunction, conjunction, or negation of predicates; bottom and top are false and true, respectively. An SMT solver can decide equality and subset constraints. This approach has been demonstrated to be effective for very large character sets in the work on symbolic finite automata [20].

The rest of this paper is generic with respect to the choice of an effective boolean algebra.

4 Antimirov's Algorithm for Checking Containment

Given two regular expressions r, s, the *containment problem* asks whether $r \sqsubseteq s$. This problem is decidable using standard techniques from automata theory: construct a deterministic finite automaton for $r\&!s$ and check it for emptiness. The drawback of this approach is the expensive construction of the automaton. In general, this expense cannot be avoided because problem is PSPACE-complete [10, 11, 15].

Antimirov [1] proposed an algorithm for deciding containment of standard regular expressions (without intersection and negation) that is based on rewriting of inequalities. His algorithm has the same asymptotic complexity as the automaton construction, but it can fail early and is therefore better behaved in practice. We phrase the algorithm in terms of Brzozowski derivatives to avoid introducing Antimirov's notion of partial derivatives.

▶ **Theorem 1** (Containment [1, Proposition 7(2)]). *For regular expressions r and s,*

$$r \sqsubseteq s \Leftrightarrow (\forall u \in \Sigma^*)\ \partial_u(r) \sqsubseteq \partial_u(s).$$

Antimirov's algorithm applies this theorem exhaustively to an inequality $r \mathrel{\dot\sqsubseteq} s$ (i.e., a proposed containment) to generate all pairs $\partial_u(r) \mathrel{\dot\sqsubseteq} \partial_u(s)$ of iterated derivatives until it finds a contradiction or saturation. More precisely, Antimirov defines a *containment calculus* $\mathcal{CC}$ which works on sets S of atoms, where an atom is either an inequality $r \mathrel{\dot\sqsubseteq} s$ or a boolean constant *true* or *false*. It consists of the rule CC-Disprove which infers *false* from a trivially inconsistent inequality and the rule CC-Unfold that applies Theorem 1 to generate new inequalities.

$$\textsc{CC-Disprove}\ \frac{\nu(r) \wedge \neg\nu(s)}{r \mathrel{\dot\sqsubseteq} s \vdash_{\mathcal{CC}} \mathit{false}} \qquad \textsc{CC-Unfold}\ \frac{\nu(r) \Rightarrow \nu(s)}{r \mathrel{\dot\sqsubseteq} s \vdash_{\mathcal{CC}} \{\partial_a(r) \mathrel{\dot\sqsubseteq} \partial_a(s) \mid a \in \Sigma\}}$$

An inference in the calculus for checking whether $r_0 \sqsubseteq s_0$ is a sequence $S_0 \vdash_{\mathcal{CC}} S_1 \vdash_{\mathcal{CC}} S_2 \vdash_{\mathcal{CC}} \dots$ where $S_0 = \{r_0 \mathrel{\dot\sqsubseteq} s_0\}$ and S_{i+1} is an extension of S_i by selecting an inequality in S_i and adding the consequences of applying one of the $\mathcal{CC}$ rules to it. That is, if $r \mathrel{\dot\sqsubseteq} s \in S_i$ and $r \mathrel{\dot\sqsubseteq} s \vdash_{\mathcal{CC}} S$, then $S_{i+1} = S_i \cup S$.

Antimirov argues [1, Theorem 8] that this algorithm is sound and complete by proving (using Theorem 1) that $r \sqsubseteq s$ does not hold if and only if a set of atoms containing *false* is derivable from $r \mathrel{\dot\sqsubseteq} s$. The algorithm terminates because there are only finitely many different inequalities derivable from $r \mathrel{\dot\sqsubseteq} s$ using rule CC-Unfold.

The containment calculus $\mathcal{CC}$ has two drawbacks. First, the choice of an inequality for the next inference step is nondeterministic. Second, an adaptation to a setting with an infinite alphabet seems doomed because rule CC-Unfold requires us to compute the derivative for infinitely many $a \in \Sigma$ at each application. We address the second drawback next.

5 Derivatives on Literals

In this section, we develop a variant of Theorem 1 that enables us to define a variant of the CC-Unfold rule that is guaranteed to add finitely many atoms, even if the alphabet is infinite. First, we observe that we may restrict the symbols considered in rule CC-Unfold to initial symbols of the left hand side of an inequality.

▶ **Definition 2** (First). Let $\mathsf{first}(r) := \{a \mid aw \in [\![r]\!]\}$ be the set of initial symbols derivable from regular expression r.

Clearly, $(\forall a \in \Sigma)\ \partial_a(r) \sqsubseteq \partial_a(s)$ iff $(\forall b \in \mathsf{first}(r))\ \partial_b(r) \sqsubseteq \partial_b(s)$ because $\partial_b(r) = \emptyset$ for all $b \notin \mathsf{first}(r)$. Thus, CC-Unfold does not have to consider the entire alphabet, but unfortunately $\mathsf{first}(r)$ may still be an infinite set of symbols. For that reason, we propose to compute derivatives with respect to *literals* (i.e., non-empty sets of symbols) instead of single symbols. However, generalizing derivatives to literals has some subtle problems.

To illustrate these problems, let us recall the specification of the Brzozowski derivative:

$$[\![\partial_a(r)]\!] = a^{-1}[\![r]\!]$$

We might be tempted to consider the following naive extension of the derivative to a set of symbols A.

$$[\![\partial_A(r)]\!] \;=\; A^{-1}[\![r]\!] \;=\; \bigcup_{a\in A} a^{-1}[\![r]\!] = \bigcup_{a\in A} [\![\partial_a(r)]\!] \qquad \text{(wrong)}$$

However, this attempt at a specification yields inconsistent results. To see why, consider the case where $r = !s$. Generalizing from $\partial_a(!s) = !\partial_a(s)$, we might try to define $\partial_A(!s) := !\partial_A(s)$. If this definition was sensible, then (1) and (2) should yield the same results:

$$[\![\partial_A(!s)]\!] \overset{\text{(wrong)}}{=} \bigcup_{a\in A} [\![\partial_a(!s)]\!] \overset{\text{def } \partial_a}{=} \bigcup_{a\in A} \overline{[\![\partial_a(s)]\!]} \qquad (1)$$

$$[\![!\partial_A(s)]\!] \overset{\text{def } \partial_a}{=} \overline{[\![\partial_A(s)]\!]} \overset{\text{(wrong)}}{=} \overline{\bigcup_{a\in A} [\![\partial_a(s)]\!]} \overset{\text{de Morgan}}{=} \bigcap_{a\in A} \overline{[\![\partial_a(s)]\!]} \qquad (2)$$

However, we obtain a contradiction: with $A = \{a, b\}$ and $s = a{\cdot}a{+}b{\cdot}b$, (1) yields Σ^* whereas (2) yields $\overline{\{a, b\}}$, which is clearly different.

5.1 Positive and Negative Derivatives

To address this problem, we introduce two types of derivative operators with respect to symbol sets. The *positive derivative* $\Delta_A(r)$ computes an expression that contains the union of all $\partial_a(r)$ with $a \in A$, whereas the *negative derivative* $\nabla_A(r)$ computes an expression contained in the intersection of all $\partial_a(r)$ with $a \in A$.

The positive and negative derivative operators are defined by mutual induction and flip at the complement operator. Most cases of their definition are identical to the Brzozowski derivative (cf. Section 2), thus we only show the cases that are different. For all literals A with $[\![A]\!] \neq \emptyset$:

$$\Delta_B(A) := \begin{cases} \epsilon, & A \sqcap B \neq \bot \\ \emptyset, & \textit{otherwise} \end{cases} \qquad \nabla_B(A) := \begin{cases} \epsilon, & \overline{A} \sqcap B = \bot \\ \emptyset, & \textit{otherwise} \end{cases}$$

$$\Delta_B(!r) := !\nabla_B(r) \qquad \nabla_B(!r) := !\Delta_B(r)$$

For single symbol literals of the form $B = \{a\}$, it holds that $\Delta_a(r) = \nabla_a(r) = \partial_a(r)$. Derivatives with respect to the empty set are defined as $\Delta_\emptyset(r) = \emptyset$ and $\nabla_\emptyset(r) = \Sigma^*$.

The following lemma states the connection between the derivative by a literal and the derivative by a symbol.

▶ **Lemma 3** (Positive and negative derivatives). *For any r and B, it holds that:*

$$[\![\Delta_B(r)]\!] \supseteq \bigcup_{a\in B} [\![\partial_a(r)]\!] \qquad\qquad [\![\nabla_B(r)]\!] \subseteq \bigcap_{a\in B} [\![\partial_a(r)]\!]$$

Proof of Lemma 3. Both inclusions are proved simultaneously by induction on r. ◀

The following examples illustrate the properties of the derivatives.

▶ **Example 4** (Positive derivative). Let r be $(a \cdot c)\&(b \cdot c)$ and let the literal $A = \{a, b\}$.

$$\Delta_A(r) = \Delta_A(a \cdot c)\&\Delta_A(b \cdot c) = c\&c \sqsupseteq \partial_a(r){+}\partial_b(r) = \emptyset{+}\emptyset$$

▶ **Example 5** (Negative derivative). Let r be $(a \cdot c){+}(b \cdot c)$ and let the literal $A = \{a, b\}$.

$$\nabla_A(r) = \nabla_A(a \cdot c){+}\nabla_A(b \cdot c) = \emptyset{+}\emptyset \sqsubseteq \partial_a(r)\&\partial_b(r) = c\&c$$

Positive (negative) derivatives yield an upper (lower) approximation to the information expected from a derivative. This approximation arises because we tried to define the derivative with respect to an *arbitrary* literal A. To obtain the precise information, we need to restrict these literals suitably to *next literals*.

5.2 Next Literals

An occurrence of a literal A in a regular expression r is *initial* if there is some $a \in \Sigma$ such that $\partial_a(r)$ reduces this occurrence. That is, the computation of $\partial_a(r)$ involves $\partial_a(A)$. Intuitively, A helps determine the first symbol of an element of $[\![r]\!]$.

▶ **Example 6** (Initial Literals).
1. Let $r_1 = \{a, b\}.a^*$. Then $\{a, b\}$ is an initial literal.
2. Let $r_2 = \{a, b\}.a^* + \{b, c\}.c^*$. Then $\{a, b\}$ and $\{b, c\}$ are initial.

Generalizing from the first example, we might be tempted to conjecture that if A is initial in r, then $(\forall a, b \in A)\ \partial_a(r) = \partial_b(r)$. However, the second example shows that this conjecture is wrong: $\{a, b\}$ is initial in r_2, but $\partial_a(r_2) = a^*$ and $\partial_b(r_2) = a^* + c^*$.

The problem with the second example is that $\{a, b\} \cap \{b, c\} \neq \emptyset$. Hence, instead of identifying initial literals of an ERE r, we define a set $\mathsf{next}(r)$ of next literals which are mutually disjoint, whose union contains $\mathsf{first}(r)$, and where the symbols in each literal yield the same derivative. In the second example, it must be that $\mathsf{next}(r_2) = \{\{a\}, \{b\}, \{c\}\}$.

It turns out that this problem arises in a number of cases when defining $\mathsf{next}(r)$ inductively. Hence, we define an operation $\bowtie$ that builds a set of mutually disjoint literals that cover the union of two sets of mutually disjoint literals.

▶ **Definition 7** (Join). Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be two sets of mutually disjoint literals.

$$\mathfrak{L}_1 \bowtie \mathfrak{L}_2 \ := \{(A_1 \sqcap A_2), (A_1 \sqcap \overline{\bigsqcup \mathfrak{L}_2}), (\overline{\bigsqcup \mathfrak{L}_1} \sqcap A_2) \mid A_1 \in \mathfrak{L}_1, A_2 \in \mathfrak{L}_2\}$$

The following lemma states the properties of the join operation.

$$\begin{aligned}
\mathsf{next}(\epsilon) &= \{\emptyset\} & \mathsf{next}(r{+}s) &= \mathsf{next}(r) \bowtie \mathsf{next}(s) \\
\mathsf{next}(A) &= \{A\} & \mathsf{next}(r{\cdot}s) &= \begin{cases} \mathsf{next}(r) \bowtie \mathsf{next}(s), & \nu(r) \\ \mathsf{next}(r), & \neg\nu(r) \end{cases} \\
& & \mathsf{next}(r^*) &= \mathsf{next}(r) \\
& & \mathsf{next}(r\&s) &= \mathsf{next}(r) \sqcap \mathsf{next}(s) \\
& & \mathsf{next}(!r) &= \mathsf{next}(r) \cup \{\textstyle\bigsqcap\{\overline{A} \mid A \in \mathsf{next}(r)\}\}
\end{aligned}$$

Figure 1 Computing next literals.

▶ **Lemma 8** (Properties of Join). *Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be non-empty sets of mutually disjoint literals.*
1. $\bigcup(\mathfrak{L}_1 \bowtie \mathfrak{L}_2) = \bigcup\mathfrak{L}_1 \cup \bigcup\mathfrak{L}_2$.
2. $(\forall A \neq A' \in \mathfrak{L}_1 \bowtie \mathfrak{L}_2)\ A \sqcap A' = \emptyset$.
3. $(\forall A \in \mathfrak{L}_1 \bowtie \mathfrak{L}_2)\ (\forall A_i \in \mathfrak{L}_i)\ A \sqcap A_i \neq \emptyset \Rightarrow A \sqsubseteq A_i$.

Proof of Lemma 8. Immediate from the definition. ◀

Figure 1 contains the definition of $\mathsf{next}(r)$. For ϵ the set of next literals consists of the empty set. The next literal of a literal A is A. The next literals of a union $r{+}s$ are computed as the join of the next literals of r and s as explained in Example 6. The next literals of a concatenation $r{\cdot}s$ are the next literals of r if r is not nullable. Otherwise, they are the join of the next literals of both operands. The next literals of a Kleene star expression r^* are the next literals of r. For an intersection $r\&s$, the set of next literals is the set of all intersections $A \sqcap A'$ of the next literals of both operands. In this case, the join operation $\bowtie$ is not needed because symbols that only appear in literals from one operand can be elided. To see this, consider $\mathsf{next}(a\&b) = \{\{a\} \sqcap \{b\}\} = \{\emptyset\}$ whereas $\{\{a\}\} \bowtie \{\{b\}\} = \{\emptyset, \{a\}, \{b\}\}$.

The set of next literals of $!r$ comprises the next literals of r and a new literal, which is the intersection of the complements of all literals in $\mathsf{next}(r)$. We might contemplate to exclude literals that contain symbols a such that $\partial_a(r)$ is equivalent to Σ^*, but we refrain from doing so because this equivalence cannot be decided with a finite set of rewrite rules [17].

The function $\mathsf{next}(r) \setminus \{\emptyset\}$ computes the equivalence classes of a partial equivalence relation $\sim$ on Σ such that equivalent symbols yield the same derivative on r. The relation is defined by $a \sim b$ if there exists $A \in \mathsf{next}(r)$ such that $a \in A$ and $b \in A$. Furthermore, the derivative by a symbol that is not part of the relation yields the empty set.

▶ **Lemma 9** (Partial Equivalence). *Let $\mathfrak{L} = \mathsf{next}(r)$.*
1. $(\forall A \in \mathfrak{L})\ (\forall a, b \in A)\ \partial_a(r) = \partial_b(r)$
2. $(\forall a \notin \bigcup\mathfrak{L})\ \partial_a(r) = \emptyset$

Proof of Lemma 9. Both proofs are by induction on r. ◀

It remains to show that $\mathsf{next}(r)$ covers all symbols in $\mathsf{first}(r)$.

▶ **Lemma 10** (First). *For all r, $\bigcup \mathsf{next}(r) \supseteq \mathsf{first}(r)$.*

Proof of Lemma 10. The proof is by induction on r. ◀

Moreover, there are only finitely many different next literals for each regular expression.

▶ **Lemma 11** (Finiteness). *For all r, $|\mathsf{next}(r)|$ is finite.*

Proof of Lemma 11. By induction on r. The base cases construct finite sets and the inductive cases build a finite number of combinations of the results from the subexpressions. ◀

Now, we put next literals to work. If we only take positive or negative derivatives with respect to next literals, then the inclusions in Lemma 3 turn into equalities. The result is that both the positive and the negative derivative, when applied to a next literal A, calculate a regular expression for the left quotient $A^{-1}[\![r]\!]$.

▶ **Theorem 12** (Left Quotient). *For all r, $A \in \mathit{next}(r) \setminus \{\emptyset\}$, and $a \in [\![A]\!]$:*

$$[\![\Delta_A(r)]\!] = [\![\nabla_A(r)]\!] = [\![\partial_a(r)]\!]$$

Proof of Lemma 12. By induction on r. ◀

Motivated by this result, we define the Brzozowski derivative for a non-empty subset A of a literal in $\mathsf{next}(r)$. This definition involves an arbitrary choice of $a \in A$, but this choice does not influence the calculated derivative according to Lemma 9, part 1.

▶ **Definition 13.** Let $A' \in \mathsf{next}(r)$. For each $\emptyset \neq A \subseteq A'$ define $\partial_A(r) := \partial_a(r)$, where $a \in A$.

▶ **Lemma 14** (Coverage). *For all a, u, and r it holds that:*

$$u \in [\![\partial_a(r)]\!] \;\Leftrightarrow\; \exists A \in \mathit{next}(r) : a \in A \wedge u \in [\![\Delta_A(r)]\!] \wedge u \in [\![\nabla_A(r)]\!]$$

Proof of Lemma 14. This result follows from Theorem 12 and Lemma 10. ◀

We conclude that to determine a finite set of representatives for all derivatives of a regular expression r it is sufficient to select one symbol a from each equivalence class $A \in \mathsf{next}(r) \setminus \{\emptyset\}$ and calculate $\partial_a(r)$. Alternatively, we may calculate $\Delta_A(r)$ or $\nabla_A(r)$ according to Theorem 12. It remains to lift this result to solving inequalities.

6 Solving Inequalities

Theorem 1 is the foundation of Antimirov's algorithm. It turns out that we can prove a stronger version of this theorem, which makes the rules CC-Disprove and CC-Unfold sound and complete and which also encompasses the soundness of the restriction to first sets.

▶ **Theorem 15** (Containment).

$$r \sqsubseteq s \;\Leftrightarrow\; (\nu(r) \Rightarrow \nu(s)) \;\wedge\; (\forall a \in \mathit{first}(r))\; \partial_a(r) \sqsubseteq \partial_a(s)$$

Proof of Theorem 15. $r \sqsubseteq s$ iff $[\![r]\!] \subseteq [\![s]\!]$ iff $(\forall w)\; w \in [\![r]\!] \Rightarrow w \in [\![s]\!]$.

Induction on w. If $w = \varepsilon$, then $\varepsilon \in [\![r]\!] \Rightarrow \varepsilon \in [\![s]\!]$ iff $\nu(r) \Rightarrow \nu(s)$. If $w = aw'$, then $a \in \mathsf{first}(r) \subseteq \mathsf{first}(s)$, $w' \in a^{-1}[\![r]\!] \subseteq a^{-1}[\![s]\!]$, which is equivalent to $\partial_a(r) \sqsubseteq \partial_a(s)$. ◀

As we remarked before, it may be very expensive (or even impossible) to construct all derivatives with respect to the first symbols, particularly for negated expressions and for large or infinite alphabets. To obtain a decision procedure for containment, we need a finite set of derivatives. Therefore, we use next literals as representatives of the first symbols and use Brzozowski derivatives *on literals* (Definition 13) on both sides.

To define the next literals of an inequality $r \dot{\sqsubseteq} s$, it would be sound to use the join of the next literals of both sides: $\mathsf{next}(r) \bowtie \mathsf{next}(s)$. However, we can do slightly better. Theorem 15 proves that the first symbols of r are sufficient to prove containment. Using the full join operation, however, would cover $\mathsf{first}(r) \cup \mathsf{first}(s)$ (by Lemma 10). Hence, we define a left-biased version of the join operator that only covers the symbols of its left operand.

▶ **Definition 16** (Left Join). Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be two sets of mutually disjoint literals.

$$\mathfrak{L}_1 \ltimes \mathfrak{L}_2 \ := \{(A_1 \sqcap A_2), (A_1 \sqcap \overline{\bigsqcup \mathfrak{L}_2}) \mid A_1 \in \mathfrak{L}_1, A_2 \in \mathfrak{L}_2\}$$

The following lemma states the properties of the left join operation.

▶ **Lemma 17** (Properties of Left Join). *Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be non-empty sets of mutually disjoint literals.*

1. $\bigcup(\mathfrak{L}_1 \ltimes \mathfrak{L}_2) = \bigcup \mathfrak{L}_1$.
2. $(\forall A \neq A' \in \mathfrak{L}_1 \ltimes \mathfrak{L}_2)\ A \sqcap A' = \emptyset$.
3. $(\forall A \in \mathfrak{L}_1 \ltimes \mathfrak{L}_2)\ (\forall A_i \in \mathfrak{L}_i)\ A \sqcap A_i \neq \emptyset \Rightarrow A \sqsubseteq A_i$.

Proof of Lemma 17. Immediate from the definition. ◀

▶ **Definition 18** (Next Literals of an Inequality). Let $r \mathrel{\dot\sqsubseteq} s$ be an inequality.

$$\mathsf{next}(r \mathrel{\dot\sqsubseteq} s) := \mathsf{next}(r) \ltimes \mathsf{next}(s)$$

Finally, we can state a generalization of Antimirov's containment theorem for EREs, where each unfolding step generates only finitely many derivatives.

▶ **Theorem 19** (Containment). *For all regular expressions r and s,*

$$r \sqsubseteq s \ \Leftrightarrow\ (\nu(r) \Rightarrow \nu(s)) \ \wedge\ (\forall A \in \mathsf{next}(r \mathrel{\dot\sqsubseteq} s))\ \partial_A(r) \sqsubseteq \partial_A(s).$$

Proof of Theorem 19. The proof is by contraposition. If $r \not\sqsubseteq s$ then $\exists A \in \mathsf{next}(r \mathrel{\dot\sqsubseteq} s)$: $\partial_A(r) \not\sqsubseteq \partial_A(s)$ or $\neg(\nu(r) \Rightarrow \nu(s))$. ◀

For $A \in \mathsf{next}(r \mathrel{\dot\sqsubseteq} s)$ define $\nabla_A(r \mathrel{\dot\sqsubseteq} s) := (\nabla_A(r) \mathrel{\dot\sqsubseteq} \Delta_A(s)) = (\partial_A(r) \mathrel{\dot\sqsubseteq} \partial_A(s))$.

▶ **Theorem 20** (Finiteness). *Let R be a finite set of regular inequalities. Define*

$$F(R) = R \cup \{\nabla_A(r \mathrel{\dot\sqsubseteq} s) \mid r \mathrel{\dot\sqsubseteq} s \in R, A \in \mathsf{next}(r \mathrel{\dot\sqsubseteq} s)\}$$

For each r and s, the set $\bigcup_{i\in\mathbb{N}} F^{(i)}(\{r \sqsubseteq s\})$ is finite.

Proof of Theorem 20. As we consider regular expressions up to similarity (cf. [4]) and $\nabla_A(r \mathrel{\dot\sqsubseteq} s) = \partial_A(r) \mathrel{\dot\sqsubseteq} \partial_A(s)$ is essentially applying the Brzozowski derivative to a pair of (extended) regular expressions, the set of these pairs is finite (because there are only finitely many dissimilar iterated Brzozowski derivatives for a regular expression [4]). ◀

These results are the basis for a complete decision procedure for solving inequalities on extended regular expressions where literals are defined via an effective boolean algebra. Figure 2 defines this procedure as a judgment of the form $\Gamma \vdash r \mathrel{\dot\sqsubseteq} s \ :\ b$, where Γ is a set of previous visited inequalities $r \mathrel{\dot\sqsubseteq} s$ with $\nu(r) \Rightarrow \nu(s)$ that are assumed to be true and $b \in \{\mathit{true}, \mathit{false}\}$. The effective boolean algebra comes into play in the computation of the next literals and in the computation of the derivatives.

Rule (Disprove) detects contradictory inequalities in the same way as Antimirov's rule CC-Disprove. Rule (Cycle) detects circular reasoning: Under the assumption that $r \mathrel{\dot\sqsubseteq} s$ holds we were not (yet) able to derive a contradiction and thus conclude that $r \mathrel{\dot\sqsubseteq} s$ holds. This rule guarantees termination because of the finiteness result (Theorem 20). The rules (Unfold-True) and (Unfold-False) apply only if $r \mathrel{\dot\sqsubseteq} s$ is neither contradictory nor in the context. A deterministic implementation would generate the literals $A \in \mathsf{next}(r \mathrel{\dot\sqsubseteq} s)$ and recursively check $\nabla_A(r \mathrel{\dot\sqsubseteq} s)$. If any of these checks returns false, then (Unfold-False) fires. Otherwise (Unfold-True) signals a successful containment proof. Theorem 19 is the basis for soundness and completeness of the unfolding rules.

(Disprove)
$$\frac{\nu(r) \qquad \neg\nu(s)}{\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ :\ \mathit{false}}$$

(Cycle)
$$\frac{r \mathrel{\dot{\sqsubseteq}} s \in \Gamma}{\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ :\ \mathit{true}}$$

(Unfold-True)
$$\frac{r \mathrel{\dot{\sqsubseteq}} s \notin \Gamma \qquad \nu(r) \Rightarrow \nu(s) \qquad \forall A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s) :\ \Gamma \cup \{r \mathrel{\dot{\sqsubseteq}} s\} \vdash \partial_A(r) \mathrel{\dot{\sqsubseteq}} \partial_A(s) \ :\ \mathit{true}}{\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ :\ \mathit{true}}$$

(Unfold-False)
$$\frac{r \mathrel{\dot{\sqsubseteq}} s \notin \Gamma \qquad \nu(r) \Rightarrow \nu(s) \qquad \exists A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s) :\ \Gamma \cup \{r \mathrel{\dot{\sqsubseteq}} s\} \vdash \partial_A(r) \mathrel{\dot{\sqsubseteq}} \partial_A(s) \ :\ \mathit{false}}{\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ :\ \mathit{false}}$$

Figure 2 Decision procedure for containment.

(Prove-Identity)
$$\Gamma \vdash r \sqsubseteq r \ :\ \mathit{true}$$

(Prove-Empty)
$$\Gamma \vdash \emptyset \sqsubseteq s \ :\ \mathit{true}$$

(Prove-Nullable)
$$\frac{\nu(s)}{\Gamma \vdash \epsilon \sqsubseteq s \ :\ \mathit{true}}$$

(Disprove-Empty)
$$\frac{\exists A \in \mathsf{next}(r) :\ A \neq \emptyset}{\Gamma \vdash r \sqsubseteq \emptyset \ :\ \mathit{false}}$$

Figure 3 Prove and disprove axioms.

▶ **Theorem 21** (Soundness). *For all regular expression r and s:*

$$\emptyset \vdash r \mathrel{\dot{\sqsubseteq}} s \ :\ \top \ \Leftrightarrow\ r \sqsubseteq s$$

Proof of Theorem 21. We prove that $\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ :\ \mathit{false}$ iff $r \not\sqsubseteq s$, for all contexts Γ where $r \mathrel{\dot{\sqsubseteq}} s \notin \Gamma$. This is sufficient because each regular inequality gives rise to a finite derivation by Theorem 20. ◀

In addition to the rules from Figure 2, we may add auxiliary rules to detect trivially consistent or inconsistent inequalities early (Figure 3 contains some examples). Such rules may be used to improve efficiency. They decide containment directly instead of unfolding repeatedly.

7 Conclusion

Antimirov's algorithm is a viable tool for proving containment of regular expressions to extended regular expressions on potentially infinite alphabets. To work effectively with such alphabets, we require that literals in regular expressions are drawn from an effective boolean algebra. As a slight difference, we work with Brzozowski derivatives instead of Antimirov's notion of partial derivative.

The main effort in lifting Antimirov's algorithm is to identify, for each regular inequality $r \mathrel{\dot{\sqsubseteq}} s$, a finite set of symbols such that calculating the derivation with respect to these symbols covers all possible derivations with all symbols. We regard the construction of the set of suitable representatives in an effective boolean algebra, embodied in the notion of next literals $\mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s)$, as a key contribution of this work.

References

1 Valentin M. Antimirov. Rewriting regular inequalities. In Horst Reichel, editor, *FCT*, volume 965 of *LNCS*, pages 116–125. Springer, 1995.

2 Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.

3 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 457–468, Rome, Italy, January 2013. ACM.

4 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

5 Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot. Partial derivatives of an extended regular expression. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *LNCS*, pages 179–191. Springer, 2011.

6 A. Ginzburg. A procedure for checking equality of regular expressions. *J. ACM*, 14(2):355–362, April 1967.

7 Victor M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1–53, 1961.

8 Phillip Heidegger, Annette Bieniusa, and Peter Thiemann. Access permission contracts for scripting languages. In John Field and Michael Hicks, editors, *Proc. 39th ACM Symp. POPL*, pages 111–122, Philadelphia, USA, January 2012. ACM Press.

9 John Edward Hopcroft and Richard Manning Karp. A linear algorithm for testing equivalence of finite automata. Technical report, Cornell University, 1971.

10 Harry B. Hunt III, Daniel J. Rosenkrantz, and Thomas G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *J. Comput. Syst. Sci.*, 12(2):222–268, 1976.

11 Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

12 Matthias Keil and Peter Thiemann. Efficient dynamic access analysis using JavaScript proxies. In *Proceedings of the 9th Symposium on Dynamic Languages*, DLS'13, pages 49–60, New York, NY, USA, 2013. ACM.

13 Matthias Keil and Peter Thiemann. Symbolic solving of regular expression inequalities. Technical report, Institute for Computer Science, University of Freiburg, 2014.

14 Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *Electronic Computers, IRE Transactions on*, EC-9(1):39–47, 1960.

15 Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT (FOCS)*, pages 125–129. IEEE Computer Society, 1972.

16 Scott Owens, John H. Reppy, and Aaron Turon. Regular-expression derivatives re-examined. *J. Funct. Program.*, 19(2):173–190, 2009.

17 Valentin N. Redko. On defining relations for the algebra of regular events. *Ukrain. Mat.*, 16:120–126, 1964.

18 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968.

19 Gertjan van Noord and Dale Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.

20 Margus Veanes. Applications of symbolic finite automata. In Stavros Konstantinidis, editor, *CIAA*, volume 7982 of *Lecture Notes in Computer Science*, pages 16–23, Halifax, NS, Canada, 2013. Springer.

21 Bruce W. Watson. Implementing and using finite automata toolkits. *Nat. Lang. Eng.*, 2(4):295–302, December 1996.

Solving the Stable Set Problem in Terms of the Odd Cycle Packing Number

Adrian Bock[1], Yuri Faenza[1], Carsten Moldenhauer[1], and Andres Jacinto Ruiz-Vargas[2]

1 DISOPT, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
2 DCG, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

The classic *stable set* problem asks to find a maximum cardinality set of pairwise non-adjacent vertices in an undirected graph G. This problem is NP-hard to approximate with factor $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [10, 24], where n is the number of vertices, and therefore there is no hope for good approximations in the general case. We study the stable set problem when restricted to graphs with bounded *odd cycle packing* number $\operatorname{ocp}(G)$, possibly by a function of n. This is the largest number of vertex-disjoint odd cycles in G. Equivalently, it is the logarithm of the largest absolute value of a sub-determinant of the edge-node incidence matrix A_G of G. Hence, if A_G is totally unimodular, then $\operatorname{ocp}(G) = 0$. Therefore, $\operatorname{ocp}(G)$ is a natural distance measure of A_G to the set of totally unimodular matrices on a scale from 1 to $n/3$.

When $\operatorname{ocp}(G) = 0$, the graph is bipartite and it is well known that stable set can be solved in polynomial time. Our results imply that the odd cycle packing number indeed strongly influences the approximability of stable set. More precisely, we obtain a polynomial-time approximation scheme for graphs with $\operatorname{ocp}(G) = o(n/\log n)$, and an α-approximation algorithm for any graph where α smoothly increases from a constant to n as $\operatorname{ocp}(G)$ grows from $O(n/\log n)$ to $n/3$. On the hardness side, we show that, assuming the exponential-time hypothesis, stable set cannot be solved in polynomial time if $\operatorname{ocp}(G) = \Omega(\log^{1+\varepsilon} n)$ for some $\varepsilon > 0$. Finally, we generalize a theorem by Györi *et al.* [9] and show that graphs without odd cycles of small weight can be made bipartite by removing a small number of vertices. This allows us to extend some of our above results to the *weighted* stable set problem.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases stable set problem, independent set problem, approximation algorithms, odd cycle packing number, maximum subdeterminants

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.187

1 Introduction

The *stable* or *independent set* problem is fundamental in combinatorial optimization. It is as follows: Given an undirected graph $G = (V, E)$, find a subset $S \subseteq V$ of maximum cardinality such that no two vertices in S are adjacent. Stable sets occur naturally when pairwise conflicts of choice have to be respected. It is among the very first combinatorial optimization problems that were shown to be NP-hard [14], and a showcase for the field of *hardness of approximation*. There is theoretical evidence that the stable set problem is extremely hard to solve: A celebrated result of Håstad [10], derandomized by Zuckerman [24], shows that unless $P = NP$, there does not exist a $\left(n^{1-\varepsilon}\right)$-approximation for the stable set problem, where n is the number of vertices of G and $\varepsilon > 0$ is any fixed constant (here and throughout the paper, n denotes the number of vertices of the graph in analysis). From a parameterized perspective, the stable set problem is $W[1]$-hard (see e.g. [4]), *i.e.*, there is no polynomial time algorithm even if the cardinality of the optimal solution is considered as a fixed parameter.

When restricted to special graph classes the problem becomes more tractable. For instance, it is polynomial time solvable in perfect graphs and claw-free graphs [5, 8, 19], while it has a PTAS e.g. on planar graphs [2]. Foremost, it can be solved efficiently on bipartite graphs. In this case, the edge-node incidence matrix

$$A_G(e,u) = \begin{cases} 1 & \text{if } u \text{ is incident to } e, \\ 0 & \text{otherwise,} \end{cases}$$

is *totally unimodular*, see, e.g. [20]. By the Hoffman-Kruskal theorem [12], all the vertices of the natural linear programming relaxation for the stable set problem

$$\max \left\{ \sum_{v \in V} x_v : A_G\, x \leq \mathbf{1}, \quad x \geq \mathbf{0} \right\}$$

are then integral, and one can resort to algorithms for linear programming to efficiently solve the problem, also in its weighted variant. A 0/1-matrix is totally unimodular if the largest absolute value of any of its sub-determinants is at most 1. The guiding questions of our paper are:

> How well can the stable set problem be approximated if A_G is not totally unimodular? Is it possible to parametrize the approximability by the largest absolute value Δ of a sub-determinant of A_G?

Recall that $\Delta = 1$, i.e. the totally unimodular case, is equivalent to G having no odd cycles. Δ maintains a neat combinatorial interpretation also when it is greater than 1. Define the *odd cycle packing number* of G to be the cardinality of a maximum family of vertex-disjoint odd cycles of G. We denote it by $\text{ocp}(G)$, omitting the dependence on G when it is clear from the context. Note that $\text{ocp}(G) \leq n/3$ for all graphs G. As observed in [7], for a graph G with edge–node incidence matrix A_G, the largest absolute value of a sub-determinant is $\Delta = 2^{\text{ocp}}$. Hence upper bounding Δ is tantamount to upper bounding the odd cycle packing number of G.

Our contribution. We show that the stable set problem can be very well approximated if $\text{ocp}(G) = o(n/\log n)$. Beyond this, the approximation guarantee smoothly approaches the hardness result from [10] as ocp approaches $n/3$.

▶ **Theorem 1.** *The stable set problem on a graph G admits the following approximation algorithms:*

- *Polynomial time approximation scheme (PTAS) if $ocp(G) = o(n/\log n)$;*
- *$O\left(n^{1/p}\right)$-approximation if $ocp(G) \leq \frac{n}{2\delta p - 1}$ for p integer (possibly a function of n) and any constant $\delta > 1$.*

Surprisingly, these approximation guarantees can be obtained using simple greedy algorithms. Theorem 1 implies that stable set is fixed-parameter tractable (with the parameter being the size of the maximum stable set) when $\text{ocp} = o(n/\log n)$, see Corollary 12, and that graphs where the hardness [10] is achieved can essentially be partitioned into triangles.

Before discussing each of the two algorithms of Theorem 1 separately, let us remark that both can be executed and will output a certificate of the approximation guarantee even without the knowledge of the ocp of the input graph. The first algorithm runs in polynomial time on graphs with ocp $= o(n/\log n)$. However, it might run in super-polynomial time on graphs with ocp $= \Omega(n/\log n)$. The second algorithm always runs in polynomial time.

In order to make the first algorithm robust against malicious input, it would be sufficient to find the ocp of the input graph. Unfortunately, computing the odd cycle packing number of a graph is NP-hard and inapproximable up to $\log^{1/2-\varepsilon} n$ for all $\varepsilon > 0$ (see the discussion in [3]). The best-known approximation algorithm has only ratio $\sqrt{n}$ [15]. But, we can use the following weak separation theorem to distinguish between graphs with $\text{ocp}(G) = \Omega(n/\log n)$ and graphs with $\text{ocp}(G) = o(n/\log^2 n)$.

▶ **Theorem 2** (Weak separation)**.** *Let $G(V, E)$ be a graph and $1 \leq c \leq n = |V|$ with c possibly depending on n. In time $O(n^4)$ we can conclude that $ocp(G) < \sqrt{cn}$, or that $ocp(G) \geq c$, and hence distinguish between graphs with $ocp(G) = \Omega(n/\log n)$ and graphs with $ocp(G) = o(n/\log^2 n)$.*

Thus, for most inputs to the algorithms from Theorem 1 we can efficiently check if they satisfy the condition on the ocp. Theorem 2 follows from a careful analysis of a simple greedy algorithm for ocp, see Theorem 3.

Considering hardness of approximation, we show that the stable set problem over G cannot be solved in polynomial time when ocp $= \Omega(\log^{1+\varepsilon} n)$ for all $\varepsilon > 0$, unless the exponential-time hypothesis is false (see Theorem 13). Hence, under the exponential-time hypothesis, Theorem 1 is best possible for graphs with ocp $\in \Omega(\log^{1+\varepsilon}(n)) \cap o(n/\log n)$ for each $\varepsilon > 0$.

The stable set problem, parameterized by the odd cycle packing number, can also be seen as a special case of integer programming parameterized by the absolute value of a sub-determinant of the constraint matrix. In particular, our hardness result provides an example of a non-polytime solvable (under the exponential-time hypothesis) integer program when the absolute value of the maximum sub-determinants is assumed to be of order $\Omega\left(n^{\log^{\varepsilon} n}\right)$.

In the weighted version of the stable set problem, a non-negative integer weight is associated to each vertex, and the goal is to find the set of non-adjacent vertices of maximum total weight. Extending our PTAS for the unweighted case we show that the weighted stable set problem on G admits a PTAS if ocp $= O(\sqrt{\log n/\log\log n})$ (see Theorem 15).

A key ingredient in all our algorithms is an idea of Györi *et al.* [9]. If a graph does not have small odd cycles, it is possible to find a small set of vertices that can be removed from the graph in order to make it bipartite. We extend the result of [9] to the vertex weighted case which supplies the necessary machinery for the PTAS for the weighted variant. This generalization is obtained by relating odd cycles of small *weight* with the *number* of vertices to be removed to make the graph bipartite. One easily checks that a variant relating odd cycles of small *weight* with the *weight* of vertices to be removed cannot exist.

Organization of the paper. We conclude this first section settling notation and highlighting related work. Section 2 is devoted to an improved approximation of ocp and Theorem 2. Section 3 is devoted to the proof of Theorem 1. We will first outline an easy algorithm (Algorithm 2) that provides the basic ideas and yields a first approximation result. This will prove the second part of Theorem 1. Then, we will discuss ideas on how to refine the analysis to prove the first part of Theorem 1. Section 4 deals with the weighted version of both Györi *et al.*'s theorem and the PTAS for the weighted stable set problem.

1.1 Notation

Throughout this paper we consider an undirected graph $G = (V(G), E(G))$. When G is clear from the context we will denote its vertex and edge sets by V and E respectively. For each $v \in V(G)$, let $N_i(v)$ denote the i-th neighborhood in a breadth-first search from v. For $S \subseteq V$, we denote by $G[S]$ the subgraph of G induced by S. For a weighted graph G and $v \in V(G)$ we use $w(v)$ to denote the weight of v and for a subset $S \subseteq V$ define $w(S) = \sum_{v \in S} w(v)$. All weights are assumed to be non-negative integers. Most of the other notation and definitions we employ are standard, and we refer the reader to textbooks for details on graphs, approximation algorithms, fixed-parameter tractability, and big-O notation (see for instance [21], [4], and [23]).

1.2 Related work

A dual concept to packing odd cycles is finding an *odd cycle transversal*, *i.e.*, the minimum number of vertices that have to be removed to bipartize the graph. We denote it by $\mathrm{oct}(G)$. Clearly for each graph G we have $\mathrm{ocp}(G) \leq \mathrm{oct}(G)$, and there are examples of graphs, called *Escher walls* [18], where $\mathrm{ocp} = 1$ and $\mathrm{oct} = \sqrt{n}$. In some special cases there are odd cycle transversals of small size. For instance, in planar graphs, it is known that $\mathrm{oct} \leq 6\,\mathrm{ocp}$ [16]. Győri *et al.* [9] showed that oct is also small when G does not have "short" odd cycles. More formally, for each $\varepsilon > 0$ (possibly a function of n), if G has no odd cycle of length at most εn, then $\mathrm{oct}(G) \leq f(\varepsilon)$ for some function f only depending on ε. This result and our new generalization are key components of our algorithms. Both computing ocp and oct is NP-hard, even when the input graph is planar [15, 6]. However, the two problems admit a $\sqrt{n}$ [15] and $\sqrt{\log n}$ [1] approximation, respectively.

The *exponential-time hypothesis* (ETH) [13] states that, for each $k \geq 3$, $k-$SAT cannot be decided with a sub-exponential time algorithm. An algorithm runs in *sub-exponential time* if it takes time $2^{o(n)}$ where n is the size of the instance. Impagliazzo *et al.*[13] also show that the ETH implies that there is no sub-exponential time algorithm for the stable set problem.

In the rest of the paper we will often use the fact that a shortest odd cycle in a graph can be found in time $O(n^3)$. This is standard, and can be found e.g. in [17].

2 Odd cycle packing

In this section, we show that a simple greedy algorithm provides a $\min\{\mathrm{ocp}, n/\mathrm{ocp}\}$-approximation for the ocp. Hence, when $\mathrm{ocp} = \Theta(\sqrt{n})$, it matches the $O(\sqrt{n})$-approximation by Kawarabayashi and Reed [15], and in all other cases improves over it.

Algorithm 1: Greedy algorithm for ocp

```
Input: a graph $G$.
Output: a family $\mathcal{C}$ of vertex-disjoint odd cycles of $G$.

1. Set $G' = G$, $\mathcal{C} = \emptyset$.
2. WHILE $G'$ is not bipartite:
     i) Find a shortest odd cycle $C$ of $G'$.
    ii) Set $\mathcal{C} = \mathcal{C} \cup \{C\}$ and $G' = G' \setminus C$.
3. Return $\mathcal{C}$.
```

▶ **Theorem 3.** *In time $O(n^4)$ Algorithm 1 finds a* $\min\{ocp, \frac{n}{ocp}\}$*-approximation for ocp.*

Proof. Let ocp := ocp(G), $|V| = n$, $\mathcal{O}$ be the optimal solution, and $\mathcal{C} = \{C_1, \dots, C_t\}$ be the solution output by Greedy. As Algorithm 1 will find at least one odd cycle (if any), we know it is an ocp-approximation. So we only need to prove that $t\frac{n}{\text{ocp}} \geq \text{ocp}$ or, equivalently, $\text{ocp}^2 \leq tn$.

For $i = 1, \dots, t$, let $|C_i| = c_i$ and k_i be the number of cycles from $\mathcal{O}$ that intersect C_i but none of $C_1, \dots, C_{i-1}$. Note that $\sum_{i=1}^t k_i = \text{ocp}$. As all cycles from $\mathcal{O}$ contributing to k_i have length at least $|C_i| = c_i$, and as cycles from $\mathcal{O}$ are vertex-disjoint, we deduce

$$\sum_{i=1}^{t} k_i c_i \leq n. \tag{1}$$

As C_i can intersect at most $|C_i| = c_i$ vertex-disjoint odd cycles, we have

$$0 \leq k_i \leq c_i \qquad \text{for all } i. \tag{2}$$

We conclude that

$$\text{ocp}^2 = (\sum_{i=1}^{t} k_i)^2 \leq t\sum_{i=1}^{t} k_i^2 \leq t\sum_{i=1}^{t} k_i c_i \leq tn,$$

where the first inequality follows from Cauchy-Schwarz, and the others from (2) and (1) respectively. The complexity bound follows from the fact that we can find an odd cycle in a graph in time $O(|V|^3)$. ◀

In order to prove Theorem 2, it is sufficient to observe that, if the greedy algorithm outputs $\mathcal{C}$ of cardinality $t < c$, then we know $\text{ocp}(G) \leq \sqrt{tn} < \sqrt{cn}$. Otherwise, $\text{ocp}(G) \geq t \geq c$. The second part of the theorem follows by taking $c = n/\log^2 n$.

It is easy to see that Theorem 3 is essentially tight for each value of ocp. We give here the construction for $\text{ocp} = \sqrt{n}$: let G be a graph with odd cycles $C_0, C_1 \dots, C_{\sqrt{n}}$ all of length $\sqrt{n}$, such that C_0 intersects each of $C_1, \dots, C_{\sqrt{n}}$ in exactly one vertex, and $C_1 \dots, C_{\sqrt{n}}$ are pairwise vertex-disjoint. Then the greedy algorithm may pick C_0 only, while $\text{ocp}(G) = \sqrt{n}$. One easily generalizes this construction to other values of ocp.

Now let $\mathcal{C}$ be the family of all odd cycles of the graph, and let *f-ocp* be the optimum solution to the following natural LP relaxation of ocp:

$$\begin{aligned}
\max \quad & \sum_{C \in \mathcal{C}} y_C & & \text{(ocp-LP)}\\
& \sum_{C : v \in C} y_C \leq 1 & \text{for all } v \in V &\\
& y_C \geq 0 & \text{for all } C \in \mathcal{C}. &
\end{aligned}$$

With arguments very close to those used in the proof of Theorem 3 one can show the following, also improving over results from [15]. We defer details to the journal version of the paper.

▶ **Theorem 4.** *The greedy algorithm is a* $\min\{ocp, \frac{|V|}{f\text{-}ocp}\}$ *algorithm for the ocp and a* $\min\{f\text{-}ocp, \frac{|V|}{f\text{-}ocp}\}$ *for f-ocp. In particular, the integrality gap of the LP given by (ocp-LP) is bounded by* $\min\{f\text{-}ocp, \frac{|V|}{f\text{-}ocp}\}$.

3 Stable set

The main idea of all subsequent algorithms is the following. In a first step, delete *small* odd cycles. Due to the definition of small, this will only delete a small portion of the graph. If the remaining graph is bipartite, we can solve the stable set problem to optimality on this remainder. Otherwise, in a second step, we can leverage the fact that the remaining graph does not contain small odd cycles and obtain a large stable set.

The precise implementation of this second step depends on the size of the ocp. We will first show a general method that does not require to know the ocp of the input graph, *i.e.*, its runtime is independent of the ocp but the approximation guarantee depends on it (Algorithm 2). Then, assuming that ocp $= o(n/\log n)$ we will show how to improve this second step to obtain a PTAS (Algorithm 3 and 4).

3.1 A first approximation algorithm

Now, assume that G does not have an *odd* cycle of size at most $2k+1$. Lemma 5 states that G must have a large stable set (depending on k). We defer the proof to the journal version. A similar result, though with slightly different dependencies, was also obtained in [22].

▶ **Lemma 5.** *Let $k \in \mathbb{N}$ and G be a graph with no odd cycles of cardinality less or equal to $2k+1$ ($k \geq 1$). Then, there exists a stable set S of size $|S| > \frac{1}{3}n^{\frac{k}{k+1}}$ which can be found in time $O(n^{5/2})$.*

▶ **Corollary 6.** *Let G be a graph without odd cycles of cardinality less or equal to $2k+1$ ($k \geq 1$). Then Lemma 5 gives a $\left(3\, n^{1/(k+1)}\right)$-approximation algorithm to the stable set problem.*

Now, given any graph G, Algorithm 2 iteratively removes odd cycles of length up to some fixed value $2k+1$, and then applies Lemma 5 to obtain a large stable set. The optimal k will depend on the odd cycle packing number of the graph. Since we want to apply it to graphs whose ocp is not known a priori, we run the algorithm for all possible values of k. Let us remark that the approximation guarantee of Algorithm 2 is in terms of the ocp. However, we can compute an upper bound on the effective guarantee without knowing the ocp. This is because the bound on the approximation guarantee compares the obtained solution with the size of the entire graph.

Algorithm 2: Approximation algorithm for general ocp

Input: a graph G.
Output: a stable set of G.

```
1. For $k = 1, \ldots, \lfloor \frac{n-1}{2} \rfloor$ do:
     i) $G' = G$, $S_k = \emptyset$.
    ii) While there exists an odd cycle of cardinality at most $2k+1$ in $G'$, delete it from $G'$.
   iii) If $G'$ is empty, choose any vertex $v$ in $G$ and set $S_k = \{v\}$.
    iv) Else apply Lemma 5 to $G'$ and obtain $S_k$.
2. Return the set $S_k$ of maximum cardinality.
```

▶ **Theorem 7.** *Algorithm 2 has approximation guarantee*

$$\begin{cases} n & \text{if } ocp = \frac{n}{3} \\ 3n(n-(2p+1)ocp)^{\frac{1}{p+1}-1} & \text{if } ocp < \frac{n}{2p+1} \text{ for some } p \in \mathbb{N} \end{cases}$$

for the stable set problem and runs in time $O\left(n^5\right)$.

Proof. There are $\Theta(n)$ iterations. Finding a shortest odd cycle takes $O(n^3)$ time. Hence, the deletion step in each iteration takes at most $O(n^4)$ time. The application of Lemma 5 takes $O(n^{5/2})$ time. Overall, Algorithm 2 runs in time $O(n^5)$.

We now prove the approximation guarantee. If G' is empty at the end of step ii), the guarantee is clearly n. Otherwise, at most $(2k+1)$ocp vertices have been deleted. Hence, if $(2k+1)\text{ocp} < n$, then G' has at least $n-(2k+1)$ocp vertices and no odd cycle of cardinality at most $2k+1$. Therefore, the application of Lemma 5 yields a stable set of size at least $\frac{1}{3}\left(n-(2k+1)\text{ocp}\right)^{\frac{k}{k+1}}$. This implies an approximation guarantee of

$$n/\left(\frac{1}{3}\left(n-(2k+1)\text{ocp}\right)^{\frac{k}{k+1}}\right) = 3n\left(n-(2k+1)\text{ocp}\right)^{\frac{1}{k+1}-1},$$

under the condition that $(2k+1)\text{ocp} < n$. Given that $\text{ocp} < \frac{n}{2p+1}$ we obtain the claimed result with $k = p$. ◀

Note that this provides a smooth transition when ocp approaches $n/3$. This is formally stated in the next corollary and establishes the second part of Theorem 1 (the difference in the exponent is due to the offset of p in the bound on the ocp).

▶ **Corollary 8.** *Let G be a graph with $ocp(G) \le \frac{n}{2\delta p+1}$ for a strictly positive integer p (possibly a function of n) and a constant $\delta > 1$. Then, Algorithm 2 yields a $O\left(n^{1/(p+1)}\right)$-approximation.*

Proof. Since $\delta > 1$ we have

$$\text{ocp} \le \frac{n}{2\delta p+1} < \frac{n}{2p+1}.$$

Thus, we can use the result from Theorem 7 and obtain a guarantee of

$$3n(n-(2p+1)\text{ocp})^{-\frac{p}{p+1}} \le 3n\left(n\left(1-\frac{2p+1}{2\delta p+1}\right)\right)^{-\frac{p}{p+1}} = 3n^{1/(p+1)}\left(\frac{2\delta p+1}{2p(\delta-1)}\right)^{\frac{p}{p+1}}.$$

Now, observe that $\frac{1}{2} \le \frac{p}{p+1} \le 1$ and $\left(\frac{2\delta p+1}{2p(\delta-1)}\right) \le \frac{\left(\delta+\frac{1}{2p}\right)}{\delta-1} \le \frac{\delta+1}{\delta-1}$. The claim immediately follows. ◀

Let us remark that Corollary 8 gives a $O(\sqrt{n})$-approximation for graphs that have $\text{ocp} \le \frac{n}{3+\delta}$ for small constant $\delta > 0$. Note that in particular triangle-free graphs fall into this category and the best known approximation algorithm for these graphs has a guarantee of $O(\sqrt{n})$ [11]. Further, if $\text{ocp} = O\left(\frac{n}{\log n}\right)$ we obtain a constant factor approximation. We will see next that for smaller ocp we can even obtain an approximation scheme.

3.2 A PTAS for $\text{ocp} = o\left(n/\log n\right)$

We now restrict our attention to graphs G with $\text{ocp} = o\left(n/\log n\right)$, *i.e.*, we assume that there exists a function $f(n) \in o(1)$ such that the input graph G on n vertices has $\text{ocp}(G) \le f(n)\frac{n}{\ln n}$. We give a PTAS for the stable set problem in these graphs. Again, the algorithm does not require the knowledge of the ocp and hence can be run on any graph. However, the *existence* of such a function f is required to prove polynomial runtime.

The main new ingredient in this algorithm is to find a small odd cycle transversal X, *i.e.*, a small set of vertices whose removal bipartizes the graph. We use a result by Györi *et al.* [9] that gives a bound on $|X|$ that is, in a certain sense, independent of the size of the graph. Given the right parameter settings, we can ensure that $|X|$ is small and can therefore be removed from the graph. Solving stable set to optimality on the remaining bipartite graph then yields the solution.

We start by restating the theorem of Györi *et al.* [9]. The proof in [9] contains a minor, non-fatal error. We restate a correct version in Theorem 9 with adapted constants. The proof is constructive and can be turned into a polytime algorithm. We do not outline the proof of Theorem 9 here since it follows (albeit with new constants) also from the weighted version in Theorem 14.

▶ **Theorem 9** (from [9]). *Let $0 < \varepsilon \leq 1$ (possibly a function of n) and assume there exists no odd cycle of cardinality smaller than εn in G. Then, there is an odd cycle transversal X of G of size $|X| \leq \frac{48}{\varepsilon} \ln \frac{5}{\varepsilon}$ that can be found in $O(n^4)$ time.*

Note that the bound in Theorem 9 is only useful for $\varepsilon = \Omega(\log n/n)$. It will turn out later that this is the reason for the limitation of our PTAS to graphs with $\text{ocp} = o(n/\log n)$.

Algorithm 3 is a technical routine that will guarantee the existence of stable sets of a certain size, given an upper bound b on the ocp, and a parameter c that will depend on the approximation factor of the PTAS.

Algorithm 3: Subroutine for computing a large stable set

Input: graph G, parameters $b \geq c > 0$.
Output: a stable set of G.

```
1. Set $\varepsilon = \frac{c}{b}$, and let $G' = G$.
2. While there exists an odd cycle of cardinality smaller than $\varepsilon|V(G)|$ in $G'$, remove it from $G'$.
3. If $G'$ is not bipartite, find an odd cycle transversal $X$ (Theorem 9) and remove it from $G'$.
4. Solve the stable set problem in $G'$ and return the solution.
```

▶ **Lemma 10.** *Let G be a graph, and b and c chosen such that $b \geq ocp(G) > 0$ and $0 < c \leq b$. Then, Algorithm 3 applied to G, b, and c finds in time $O(n^4)$ a stable set of size at least*

$$\frac{1}{2}\left((1-c)n - \frac{48\,b}{c}\ln\frac{5\,b}{c}\right).$$

Proof. The algorithm removes at most ocp odd cycles of size at most εn in Step 2. Hence, G' has at least $n - \text{ocp}\ \varepsilon n \geq n - b\frac{c}{b}n = (1-c)n$ vertices. The claim now follows using the bound on $|X|$ from Theorem 9 and the fact that every bipartite graph admits a stable set on at least half of its vertices.

Finding a shortest odd cycle in G' takes $O(|V(G')|^3) \subseteq O(n^3)$ time. Since the removal of an odd cycle means the removal of at least three vertices, this procedure is only needed $O(n)$ times. Hence, Step 2 takes $O(n^4)$ time. Step 3 takes $O(n^4)$ time (Theorem 9). Solving maximum stable set in G' takes $O(n^{5/2})$ time [21, Cor. 19.3a]. In fact, the proof requires only one side of the bipartition which can be found in $O(n^2)$. Concluding, the runtime of Algorithm 3 is $O(n^4)$. ◀

```
Algorithm 4: PTAS for the unweighted case when ocp = $o(n/\log n)$
Input: graph $G$ and a parameter $\delta > 0$.
Output: a stable set of $G$.
1. Set $c = \frac{\delta}{3+2\delta}$ and $b = \frac{c^2 n}{50 \ln n}$.
2. Run Algorithm 3 with $b$ and $c$ and obtain a solution $S$. Let $\mathcal{C}$ denote the union of the cycles
   removed in Step 2 of Algorithm 3 and $X$ denote the set found in Step 3 of Algorithm 3.
3. If $\frac{1}{2}|\mathcal{C}| + |X| \leq \delta|S|$, return $S$.
4. Otherwise, solve the maximum stable set problem by complete enumeration.
```

▶ **Theorem 11** (Polynomial time approximation scheme). *Let $1 \geq \delta > 0$ and G be a graph with ocp$(G) = h(n)$, where $h(n) = o\left(\frac{n}{\log n}\right)$. Algorithm 4 is a $(1+\delta)$ approximation for the maximum stable set problem on G and runs in time $p(\delta) + O(n^4)$ for an appropriate function p depending only on δ.*

Proof. If Algorithm 4 returns a solution in Step 4, we obtain the optimal solution. Hence, assume that Algorithm 4 returns a solution in Step 3. We derive an upper bound on the size of the optimal solution OPT(G). Let G' denote the bipartite graph after the application of Step 3 of Algorithm 3. Recall that $G = G' \cup X \cup \mathcal{C}$. In each cycle of $\mathcal{C}$, the optimal solution can take at most half of the vertices. Note that S is the optimal solution on the bipartite graph G' (Step 4 of Algorithm 3). Hence,

$$\mathrm{OPT}(G) \leq \mathrm{OPT}(G') + |X| + \frac{1}{2}|\mathcal{C}| = |S| + |X| + \frac{1}{2}|\mathcal{C}| \leq (1+\delta)|S|.$$

Since ocp$(G) = o(n/\log n)$ there exists a function $f \in o(1)$ with ocp$(G) \leq f(n)\frac{n}{\ln n}$. Let N be the constant

$$N = \min\left\{m \in \mathbb{N} : \frac{c^2 m}{50 \ln m} \geq 1 \text{ and } f(m') \leq \frac{c^2}{50} \text{ for all } m' \geq m\right\}.$$

We now prove that for $n \geq N$ the condition in Step 3 of Algorithm 4 will be satisfied. Hence, the algorithm only uses complete enumeration for constant size graphs, where the constant only depends on δ when we fix the function h in the statement of the theorem. Otherwise, it relies on Algorithm 3 which runs in time $O(n^4)$. This concludes the analysis of the running time.

Therefore, assume $n \geq N$. By the definition of N and b we have $b \geq 1$ and $b = \frac{c^2 n}{50 \ln n} \geq f(n)\frac{n}{\ln n} \geq$ ocp(G). Thus, b is a valid upper bound for ocp. The size of each cycle in $\mathcal{C}$ is upper bounded by εn. Therefore $|\mathcal{C}| \leq \text{ocp}\,\varepsilon\, n \leq b\, \varepsilon\, n = c\, n$. Furthermore, we have from Theorem 9 and the setting of b that

$$|X| \leq \frac{48b}{c} \ln\left(\frac{5b}{c}\right) \leq c\, n \frac{1}{\ln n} \ln\left(\frac{n}{\ln n}\right) \leq c\, n.$$

Plugging this into the bound of Lemma 10 gives $|S| = \mathrm{OPT}(G') \geq \frac{1}{2}\left((1-c)n - c\, n\right) = \frac{1}{2}\left(1-2c\right)n$. Finally, using $c = \frac{\delta}{3+2\delta}$ we obtain

$$\frac{|X| + \frac{1}{2}|\mathcal{C}|}{|S|} \leq \frac{cn + \frac{1}{2}cn}{\frac{1}{2}\left(1-2c\right)n} = \frac{3c}{1-2c} = \delta.$$

Thus, if $n \geq N$, Algorithm 4 returns a solution in Step 3. ◀

3.3 Fixed-parameter tractability

Algorithm 4 is in fact an *efficient* PTAS (see e.g. [4]), for its running time is at most $p(\delta) + O(n^4)$, for appropriate function p. Using standard arguments (see [4, Theorem 4.6]) we can use Algorithm 4 to conclude the following.

▶ **Corollary 12.** *Let $h : \mathbb{N} \to \mathbb{N}$ be a function such that $h(n) = o(n/\log n)$. Then, the stable set problem is FPT in the class of graphs with ocp $= h(n)$. More precisely, for a graph G in this class it can be solved in time $f(OPT) + n^4$, where OPT is the maximum stable set and f is an appropriate function depending only on OPT.*

3.4 Hardness

▶ **Theorem 13.** *If the ETH is true, then for each $\varepsilon > 0$, the maximum stable set problem cannot be solved in polynomial time for graphs with ocp $= \Omega((\log n)^{1+\varepsilon})$.*

A sketch of the proof is as follows: given a graph G, we can "blow it up" in order to obtain a new graph G' with much more vertices, but the same ocp. In particular, the ocp will be much smaller with respect to the number of vertices of the graph. Then, we show that solving the stable set problem for G' in polynomial time implies that we can solve the stable set problem for G in sub-exponential time, contradicting the ETH. We defer the details to the journal version.

4 Weighted stable set

4.1 Graphs without odd cycles of small weight are almost bipartite

In this section, we show that a graph without odd cycles of small weight has an odd cycle transversal of small cardinality. This generalizes Theorem 9 by Györi *et al.* [9]. The proof of Theorem 14 follows the same framework of the original: we iteratively find and delete "small" neighborhoods of vertices from a "small" cycle. However, the presence of weights implies some difficulties that are non-trivial to overcome. Let us mention two here: first, unlike in the unweighted case, not all vertices from a "small" cycle are a suitable starting point for this procedure. Second, it is not clear a priori how to define weighted neighborhoods appropriately. We postpone the full proof to the journal version. The weighted version also provides a proof of the unweighted case with different constants, but the same dependency on ε.

▶ **Theorem 14.** *Let G be a graph with n vertices and node weights w. Let $0 < \varepsilon \leq 1$ and assume there exists no odd cycle C of G with $w(C) < \varepsilon w(V)$. Then, there exists an odd cycle transversal X of G of size $|X| \leq \frac{96}{\varepsilon} \ln \frac{10}{\varepsilon}$ that can be found in $O(n^4 \log(w(V)))$ time.*

4.2 A PTAS for ocp $= O(\sqrt{\frac{\log n}{\log \log n}})$

In this section, we present a PTAS for the *weighted* stable set problem. Note that this time we need to know the ocp of the graph in advance. We only sketch the proof here and defer it to the journal version.

Similarly to the unweighted case, we iteratively remove odd cycles of small *weight* until we can find an odd cycle transversal X of small cardinality, whose existence is guaranteed by Theorem 14. There are two obstacles to overcome. First, the weight of the set X can be very high, so we cannot afford to simply remove it as in the unweighted case. Second, the maximum weight of a stable set in the graph is not necessarily a constant fraction of the total weight of the graph. Thus we have to ensure that the total weight of the odd cycles that we remove is small with respect to the weight of the optimum solution.

We overcome the first obstacle by enumerating all possible stable sets on X. The second obstacle can be dealt with via the following observation. The input graph $G = (V, E)$ can be partitioned into ocp many odd cycles and one bipartite subgraph. By the pigeonhole principle, one of these subgraphs has weight at least $\frac{1}{\text{ocp}+1}w(V)$. Since the maximum stable set of an odd cycle C is at least $\frac{1}{3}w(C)$, we conclude that the maximum weight stable set of G has weight at least $\frac{1}{3\text{ocp}+3}w(V) \geq \frac{1}{6\text{ocp}}w(V)$. The combination of those two difficulties limits the applicability of our PTAS to graphs with $\text{ocp} = O(\sqrt{\log n/\log\log n})$.

▶ **Theorem 15.** *Algorithm 5 is a PTAS for the weighted stable set problem if ocp =* $O\left(\sqrt{\frac{\log n}{\log\log n}}\right)$.

Algorithm 5: PTAS for the weighted stable set problem in graphs with $ocp = O\left(\sqrt{\frac{\log n}{\log\log n}}\right)$

Input: graph G, vertex weights $w : V \to \mathbb{N}$, a parameter $\delta > 0$.
Output: a stable set of G of weight at least $\frac{1}{1+\delta}$ times the optimal value.

```
1. Set $\varepsilon = \frac{\delta}{6(1+\delta)\text{ocp}^2}$ for the remainder of the algorithm and use $G' = G$ as a copy.
2. While there exists an odd cycle of weight at most $\varepsilon w(G')$ in $G'$, remove it from $G'$.
3. Apply Theorem 14 to find an odd cycle transversal $X$ of $G'$.
4. Compute the maximum weight stable set in $G'$:
   - For each stable set $\bar{S} \subseteq X$, compute the maximum weighted stable set $S'$ in $G' \setminus (X \cup N(\bar{S}))$.
   - Return $\bar{S} \cup S'$ with maximum weight.
```

Acknowledgments. We thank Friedrich Eisenbrand for attracting our attention to the problem and for stimulating discussions. Yuri Faenza was supported by the German Research Foundation (DFG) within the Priority Programme 1307 Algorithm Engineering. Andres J. Ruiz-Vargas was supported by the Swiss National Science Foundation grants 200020-144531 and 200021-137574.

References

1 A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ Approximation Algorithms for Min UnCut, Min 2CNF Deletion, and Directed Cut Problems. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC'05, pages 573–581, New York, NY, USA, 2005.

2 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, January 1994.

3 M. Di Summa, F. Eisenbrand, Y. Faenza, and C. Moldenhauer. On largest volume simplices and sub-determinants. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, California, USA, January 04-06, 2015*, 2015.

4 R. G. Downey and M. R. Fellows. *Parametrized Complexity*. Springer-Verlag New York, Inc., New York, NY, USA, 1999.

5 Y. Faenza, G. Oriolo, and G. Stauffer. Solving the weighted stable set problem in claw-free graphs via decomposition. *Journal of the ACM*, 61(4), 2014.

6 S. Fiorini, N. Hardy, B. Reed, and A. Vetta. Approximate min-max relations for odd cycles in planar graphs. *Mathematical Programming*, 110(1, Ser. B):71–91, 2007.

7 J. W. Grossman, D. M. Kulkarni, and I. E. Schochetman. On the minors of an incidence matrix and its smith normal form. *Linear Algebra and its Applications*, 218:213 – 224, 1995.

8 M. Grötschel, L. Lovász, and A. Schrijver. Stable sets in graphs. In *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*, pages 272–303. Springer Berlin Heidelberg, 1988.

9 E. Györi, A. V. Kostochka, and T. Luczak. Graphs without short odd cycles are nearly bipartite. *Discrete Mathematics*, 163(1–3):279–284, 1997.

10 J. Håstad. Clique is hard to approximate within $n(1-\varepsilon)$. In *Acta Mathematica*, pages 627–636, 1996.

11 M. M. Halldórsson. Approximations of independent sets in graphs. In Klaus Jansen and José Rolim, editors, *Approximation Algorithms for Combinatiorial Optimization*, volume 1444 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 1998.

12 A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In *Linear inequalities and related systems*, Annals of Mathematics Studies, no. 38, pages 223–246. Princeton University Press, Princeton, N. J., 1956.

13 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512 – 530, 2001.

14 R. M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

15 K.-I. Kawarabayashi and B. Reed. Odd cycle packing. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC'10, pages 695–704. ACM, 2010.

16 D. Král, J.-S. Sereni, and L. Stacho. Min-max relations for odd cycles in planar graphs. *SIAM Journal on Discrete Mathematics*, 26(3):884–895, 2012.

17 B. Monien. The complexity of embedding graphs into binary trees. In *Fundamentals of Computation Theory, FCT'85, Cottbus, GDR, September 9-13, 1985*, pages 300–309, 1985.

18 B. Reed. Mangoes and blueberries. *Combinatorica*, 19:267–296, 1999.

19 N. Sbihi. Algorithme de recherche d'un stable de cardinalite maximum dans un graphe sans etoile. *Discrete Mathematics*, 29(1):53 – 76, 1980.

20 A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1998.

21 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume A. Springer, 2003.

22 J. B. Shearer. The independence number of dense graphs with large odd girth. *The Electronic Journal of Combinatorics*, 2, 1995.

23 D.B. West. *Introduction to Graph Theory.* Prentice Hall, 2000.

24 D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC'06, pages 681–690, New York, NY, USA, 2006. ACM.

Lift & Project Systems Performing on the Partial Vertex Cover Polytope*

Konstantinos Georgiou and Edward Lee†

Department of Combinatorics and Optimization, University of Waterloo
200 University Ave. W, Waterloo ON, N2L 3G1 Canada
k2georgiou@uwaterloo.ca, e45lee@uwaterloo.ca

Abstract

We study integrality gap (IG) lower bounds on strong LP and SDP relaxations derived by the Sherali-Adams (SA), Lovász-Schrijver-SDP (LS_+), and Sherali-Adams-SDP (SA_+) lift-and-project (L&P) systems for the t-Partial-Vertex-Cover (t-PVC) problem, a variation of the classic Vertex-Cover problem in which only t edges need to be covered. t-PVC admits a 2-approximation using various algorithmic techniques, all relying on a natural LP relaxation. Starting from this LP relaxation, our main results assert that for every $\epsilon > 0$, level-$\Theta(n)$ LPs or SDPs derived by all known L&P systems that have been used for positive algorithmic results (but the Lasserre hierarchy) have IGs at least $(1-\epsilon)n/t$, where n is the number of vertices of the input graph. Our lower bounds are nearly tight, in that level-n relaxations, even of the weakest systems, have integrality gap 1.

As lift-and-project systems have given the best algorithms known for numerous combinatorial optimization problems, our results show that restricted yet powerful models of computation derived by many L&P systems fail to witness c-approximate solutions to t-PVC for any constant c, and for $t = O(n)$. This is one of the very few known examples of an intractable combinatorial optimization problem for which LP-based algorithms induce a constant approximation ratio, still lift-and-project LP and SDP tightenings of the same LP have unbounded IGs.

As further motivation for our results, we show that the SDP that has given the best algorithm known for t-PVC has integrality gap n/t on instances that can be solved by the level-1 LP relaxation derived by the LS system. This constitutes another rare phenomenon where (even in specific instances) a static LP outperforms an SDP that has been used for the best approximation guarantee for the problem at hand.

Finally, we believe our results are of independent interest as they are among the very few known integrality gap lower bounds for LP and SDP 0-1 relaxations in which not all variables possess the same semantics in the underlying combinatorial optimization problem. Most importantly, one of our main contributions is that we make explicit of a new and simple methodology of constructing solutions to LP relaxations that almost trivially satisfy constraints derived by all SDP L&P systems known to be useful for algorithmic positive results (except the La system). The latter sheds some light as to why La tightenings seem strictly stronger than LS_+ or SA_+ tightenings.

1998 ACM Subject Classification G.1.6 Convex Programming, G.2.0 Combinatorial Algorithms

Keywords and phrases Partial vertex cover, combinatorial optimization, linear programming, semidefinite programming, lift and project systems, integrality gaps

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.199

* Research completed as part of the Undergraduate Research Program at the Department of Combinatorics and Optimization, University of Waterloo.
† Research supported by NSERC of Canada.

1 Introduction

Let $G = (V, E)$ be a graph on n vertices and $t \in \mathbb{N}$, with $t \leq |E|$. A subset of vertices S that are incident to at least t many edges is called a *t-partial vertex cover.* In the t-Partial-Vertex-Cover (t-PVC) optimization problem, the goal is to find a t-partial vertex cover S of minimum size. t-PVC is a tractable optimization problem whenever $t = \Theta(1)$. In the other extreme, $|E|$-PVC is exactly the classic **NP**-hard problem known as minimum Vertex-Cover (VC). As such, any hardness of approximation for VC translates to the same hardness for $|E|$-PVC. In particular, $|E|$-PVC is 1.36 and $(2 - o(1))$ hard to approximate assuming $\mathbf{P} \neq \mathbf{NP}$ [10] and the Unique Games Conjecture [18] respectively. Moreover, there exists an approximation preserving reduction from t-PVC to VC as long as $n/t = n^{\Theta(1)}$ [4]. Unlike VC, t-PVC is also known to be hard in bipartite graphs [5]. On the positive side, [16, 25, 31] have proposed 2-approximation algorithms even for the weighted version of t-PVC (see [20] for a wider family of results concerning partial covering problems). The common starting point of all these results is the standard 0-1 LP relaxation for t-PVC (see (t-PVC-LP) in Section 2.1). The best (asymptotic) approximation known for t-PVC relies on a SDP relaxation and achieves a $2 - \Omega(\log\log n/\log n)$ ratio [15].

A standard performance measure for convex-programming (LP or SDP) relaxations is the so-called integrality gap (IG), i.e. the worst possible ratio between the cost of the exact optimal solution and the cost of the relaxation. As a measure of complexity, IG upper or lower bounds are informative for two main reasons: (1) the majority of convex-programming based approximation algorithms attain an approximation ratio equal to the best provable upper bound on the IG. (2) Convex-programming relaxations can be seen as a restricted and static model of computation that can immediately witness using fractional solutions the existence of good integral and approximate solutions.

For a long series of combinatorial optimization problems, the best approximability known agrees with the IG of natural convex-programming relaxations. In contrast, all analyses for convex-programming relaxations for t-PVC [16, 25, 15] witness some integral solution with cost *sol* to the relaxation satisfying $sol \leq 2 \cdot rel + \Theta(1)$, where *rel* is the value of the relaxation. This leaves open the possibility that the IG of these relaxations is unbounded when the optimal solution has small enough cost. In fact, it was already known that the standard 0-1 relaxation (t-PVC-LP) has IG at least n/t. We establish the same IG for the SDP of [15].

However, the power of convex-programming for combinatorial optimization problems is not limited by the performance of the natural and static relaxations. A number of systematic procedures, known as lift-and-project (L&P) systems, have been proposed in order to reduce the IG of 0-1 LP relaxations $P \subseteq [0, 1]^m$ (the reader should think of P as the feasible region of a relaxation of some combinatorial problem). The seminal works of Lovász and Schrijver [23], Sherali and Adams [28], and Lasserre [21] give such systematic methods (LS, LS_+, SA, and La respectively).[1] Starting with the polytope P, each of the systems derives a sequence (hierarchy) of relaxations $P^{(r)}$ for $P \cap \{0, 1\}^m$ that are nested, preserve the integral solutions of P, and $P^{(m)}$ is exactly the integral hull of P (hence the IG of the last relaxation is 1 independently of the underlying objective). For these reasons, these systems are also known as hierarchies (of LP or SDP relaxations). More importantly, if P admits a (weak) separation oracle, then one can optimize a linear objective over the so-called level$-r$ relaxation $P^{(r)}$ of

[1] LS_+ and SA systems derive stronger relaxations than the LS system, while LS_+, SA are incomparable. La derives SDPs that are at least as srong than relaxations derived by any other system.

all methods but the La system in time $m^{O(r)}$ (the same is true also for the La system if the initial relaxation has polysize). In other words, all L&P systems constitute "parameterized" models of computation for attacking intractable combinatorial optimization problems.

There are numerous combinatorial problems for which either L&P systems have given the best approximation algorithms known (with no matching combinatorial algorithms known), or with approximation guarantees matching the best combinatorial algorithms known. We refer the reader to [8] for a relatively recent survey.

For this reason, a long line of research has been devoted in proving IG lower bounds for relaxations derived by L&P systems, while any such result is understood as strong evidence of the true inapproximability of the combinatorial problem at hand. At the same time, an α IG for level-r relaxations derived by L&P systems implies that algorithms (for a restricted yet powerful model of computation) that run in time $m^{O(r)}$ cannot witness the existence of α-approximate solutions to the combinatorial problem. It is notable that examples of integrality gaps for L&P systems that are way off from the best approximability known for a combinatorial optimization problem are quite rare.

1.1 Our contributions and comparison to previous work

To the best of our knowledge, this is the first study of integrality gap lower bounds for lift-and-project tightenings of the natural 0-1 relaxation of t-PVC. Our starting point is the standard LP relaxation (t-PVC-LP) that has been used in all 2-approximation algorithms for weighted instances. We aim to derive strong integrality gap lower bounds for level-r relaxations derived by the LS_+, SA and SA_+ systems, where r is as large as possible, and $t = O(n)$ (where n is the number of vertices in the input graph). It is worthwhile noticing that there is a number of very strong IG lower bounds known for VC in L&P systems, including IG of $2 - \epsilon$, for every $\epsilon > 0$, for level-$\Theta(n)$ LS LPs [27], level-$n^{\Theta(1)}$ SA LPs [6], level-$\Theta(\sqrt{\log / \log \log n})$ LS_+ SDPs [14], level-5 SA_+ SDPs [2], and IG of $7/6 - \epsilon$ and 1.36 for level-$\Theta(n)$ [26] and level-$n^{\Theta(1)}$ [32] La SDPs. Each of the aforementioned lower bounds imply directly the same IG lower bounds, for the same level relaxation and for the same system for (t-PVC-LP) by a straightforward reduction. Nevertheless, for the magnitude of t for t-PVC for which we establish our results (roughly speaking for $t \le n/2$), and in which the problem makes the transition from tractable to intractable, our IG lower bounds are superconstant.

The majority of our results are negative. Our motivating observations are that (a) a simple graph instance is responsible for a n/t IG of the SDP of [15] (Proposition 2), on which the best algorithm know for t-PVC is based and (b) the level-1 LP derived by the LS system (which is strictly weaker than the LS_+ and SA systems) solves the same instances exactly (Proposition 5). This is a remarkable example of a simple LP that outperforms, even in a specific instance, an SDP that has been used for the best algorithm for a combinatorial problem (the authors are not aware of another similar example). It is natural then to ask whether relaxations derived by L&P systems can witness existence of 2-approximate solutions to t-PVC. We answer this question in the negative by proving strong IG lower bounds for all L&P systems (but the La system) that have been used for positive algorithmic results. For all these systems we show that as long as $n \ge 2r + 2t + 2$, the level-r relaxations have integrality gap at least $\binom{n-2r}{2}/t \cdot n$. As an immediate corollary, we see that the integrality gap of the starting LP (which is at least n/t) remains $(1-\epsilon)\frac{n}{t}$ for level-$\Theta(n)$ LP and SDP relaxations. Our results could have also been stated as rank lower bounds of a certain knapsack-type inequality (the one certifying a good IG). Many similar results have appeared in the literature, e.g. [9, 22, 7], but they are all for polytopes that are of different structure than the partial vertex cover polytope.

The above negative results bring up another rare phenomenon; for the family of tractable combinatorial optimization problems t-PVC, for which $t = \Theta(1)$, L&P-relaxations have unbounded discrepancy. The authors are aware only of one more similar result [24]. This is in contrast to many combinatorial optimization problems, and in particular VC, for which constant-level L&P-relaxations either have integrality gaps matching the best approximability or they even solve tractable variations of the problems. Finally, due to the approximation preserving reduction from VC to t-PVC [4], when $t = n^{\Theta(1)}$, our results also imply that L&P systems applied on the t-PVC standard polytope cannot yield new insights for the **NP**-hardness inapproximability of VC.

We believe that our results are of independent interest also for two more reasons. The first reason is that relaxation (t-PVC-LP), for which we establish strong IG L&P lower bounds, is defined over two types of variables, i.e. vertex and edge variables corresponding to different semantics. IG lower bounds for L&P relaxations of such polytopes are very rare (the authors are aware only of one such result [19]). The second reason is that it is not well understood under which conditions semidefinite programming delivers better algorithmic properties than linear programming. Especially for LPs, the probabilistic interpretation of SA system (deriving the strongest LPs known), on which we elaborate below, has unified our understanding both for positive and negative results. When it comes to SDPs, one needs to employ seemingly stronger arguments that enhances the probabilistic interpretation of the systems with a geometric substance. Interestingly, with our technique for showing L&P lower bounds, we make explicit that it is possible to devise solutions to LP relaxations that satisfy many PSD conditions, almost trivially. For this we identify a generic and remarkably simple condition of solutions to LP relaxations that can fool a large family of PSD constraints (for a high level explanation of the condition see Section 1.2). We hope that this simple observation can help towards bridging our understanding for LP and SDP relaxations.

1.2 Our techniques

For our main results we employ some standard and generic techniques for constructing vector solutions for convex relaxations derived by the SA system. Then we identify a condition special to our solution that allows us to argue that the same construction is robust against SDP tightenings. Our IG instance is the unweighted clique on n vertices, which for all t, admits an optimal solution of cost 1. This IG construction suffers a decay that is proportional to $\binom{n-2r}{2}$. The decay with r is unavoidable, at a high level, due to that level-r relaxations solve accurately local subinstances induced by r many elements corresponding to variables. Since our LP relaxation has edge variables, the removal of r many edges induces a clique of $n-2r$ vertices. Since we still have $\binom{n-2r}{2}$ edges, each edge needs to be covered "on average" $t/\binom{n-2r}{2}$ fractional times. Due to the symmetry imposed in our solutions, this is also the contribution of each vertex in the objective.

Establishing the SA IG lower bound: A common and generic approach for constructing SA solutions is to use the probabilistic interpretation of the system, first introduced in [17], and that is implicit in all our arguments of Section 3. At a high level, the curse and the blessing of the SA system is that level-r solutions are convex combinations of (LP feasible) vectors that are integral in any set of r many variable-indices. These convex combinations can be interpreted as families of distributions of *feasible* integral solutions for subsets of the input instance of size-r (hence subsets of variables as well), that additionally enjoy the so-called *local-consistency* property: distributions over different subinstances should agree on the solutions of the common sub-subinstance. Designing such probability distributions over sets of indices that also enclose the support of any constraint gives automatically a solution to the level-r SA. Finding however such distributions is in general highly non trivial, especially when aiming for a big integrality gap.

The previous recipe is not directly applicable to the t-PVC polytope, as it has a defining facet that involves all edges of the input graph. This means that had we blindly tried to find families of probability distributions as described above, then we would have unavoidably defined distributions of feasible solutions in the integral hull. Our strategy is to deviate from the generic probabilistic approach, and focus first on satisfying constraints of the SA relaxation of relatively small support.

At a high level, the novelty of our approach is that we do not explicitly define locally consistent distributions of local 0-1 assignments, one for each subset of variables of bounded size, rather we achieve this implicitly. One of the advantages of our construction is that it is surprisingly simple. Specifically, we define a *global* distribution of 0-1 assignments as follows: each of the vertices is chosen in the solution independently at random, and with negligible probability, and covered edges are those incident to at least one chosen vertex.

The locally consistent distributions, that we need to associate each subset of variables A with, are obtained by restricting the global distribution onto the subinstance induced by A. This trick can be thought as a vast generalization of the so-called correction-phase (or expansion recovery) that is common to all SA lower bounds, although it is sometimes hidden in the technicalities of the proofs ([13] is a good example where the correction phase is made explicit). According to this trick, set A is effectively blown up (or "corrected") to a big enough superset $\overline{A}$ with certain structural properties. This allows for sampling almost uniformly at random over local 0-1 assignments (of variables in $\overline{A}$) that can be easily seen to induce consistent local distributions, whereas the same task seems to be impossible to be realized directly on A. Interestingly, $\overline{A}$ is the whole instance in our case.

Our global distribution has a special property that it always satisfies all linear constraints of the t-PVC polytope but the one demand-constraint, i.e. the constraint that requires t many edges to be covered. In particular, the proposed vector solution is a convex combination of exponentially many solutions in the integral hull and of the outlier all-0 vector. In fact our global distribution assigns probability $1 - o(1)$ to the latter vector, which is also responsible for the large integrality gap.

Notably, there is no generic reason to believe that such a vector solution satisfies the almost global constraint of the t-PVC polytope that involves all edges. To that end, we take advantage of the fact that we do *not* need to define feasible solutions of the whole instance in every small subinstance. This means that if presented with a small subinstance of the input graph, we are allowed in principle to cover zero edges in that subgraph with positive probability, as long as we do cover t many edges in the complement. That said, constraints of large support cannot be treated probabilistically with respect to the global distribution. Instead, we deal with such constraints almost algebraically (in contrast to the majority of SA constructions), as one would normally do for a standard LP. More specifically, we rely on the fact that when we condition on covering zero edges in a subclique of size at most $2r$, edges that do not touch this subclique are covered independently at random with significant probability compared to how many edges are left. Linearity of expectation then can prove for us that the demand constraint is indeed satisfied.

Establishing IG lower bounds for SDP hierarchies: Showing that our SA vector solution is robust against SDP tightenings is by construction very easy. The reason is that all SDP hierarchies (that have been used for positive algorithmic results), except the La system, distinguish constraints between those imposed by the starting 0-1 relaxation, and that are *always* linear, and PSD constraints that are valid for *all* 0-1 assignments (independently of the starting relaxation). As a result, any IG lower bound for strong LP relaxations that is based on a solution that comes from a global distribution of 0-1 assignments immediately translates into the same IG for a series of SDP hierarhies. A natural question that is raised is whether such global distributions of 0-1 assignments can be used to fool strong LP relaxations (and we answer this in the positive as we explain above). The second question that we raise is whether our solution is robust also against Lasserre tightenings. We answer this in the negative in Section 4.2.

2 Preliminaries

We denote by $\mathbf{1}_n$ the all-1 vector of dimension n, and we drop the subscript, whenever the dimension is clear from the context. Similarly, by all-α vector we mean the vector $\alpha\mathbf{1}$. For a fixed set of indices $[m] := \{1, \dots, m\}$, we denote by $\mathcal{P}_r$ all subsets of $[m]$ of size at most r (for the partial vertex cover polytope and for a graph $G = (V, E)$, we will use $[m] = V \cup E$). For some $y \in \mathbb{R}^{\mathcal{P}_{r+1}}$, we denote by $\mathcal{Y}$ the so-called *moment matrix* of y that is indexed by $\mathcal{P}_1$ in the rows and by $\mathcal{P}_r$ in the columns, with $\mathcal{Y}_{A,B} = y_{A\cup B}$. In other words, $\mathcal{Y} \in \mathbb{R}^{|\mathcal{P}_1|\times|\mathcal{P}_r|}$ whenever $y \in \mathbb{R}^{\mathcal{P}_{r+1}}$, whereas $\mathcal{Y}$ is a square symmetric matrix if $r = 1$. Finally, we denote by $\{\mathbf{e}_I\}_{I\in\mathcal{P}_r}$ the standard orthonormal basis of $\mathcal{P}_r$, so that $\mathcal{Y}\mathbf{e}_A$ is the column of $\mathcal{Y}$ indexed by set A.

Note: In the interest of space, we have omitted some proofs from this version of the paper; we refer the reader to [12] for a full presentation.

2.1 Problem Definition and a Natural LP Relaxation

Given an integer t, and a graph $G = (V, E)$ with vertex weights $w_i \in \mathbb{R}_+$ for each $i \in V$, t-PVC can be alternatively defined as the following optimization problem where variables $\{x_q\}_{q\in V\cup E}$ are further restricted to be integral.

$$\min \sum_{i\in V} w_i\, x_i \qquad (t\text{-PVC-LP})$$

$$\text{s.t.}\quad x_i + x_j \geq x_e, \qquad \forall e = \{i,j\} \in E \qquad (1)$$

$$\sum_{e\in E} x_e \geq t \qquad (2)$$

$$0 \leq x_q \leq 1 \qquad \forall q \in V \cup E \qquad (3)$$

Below we focus on uniform instances, in which $w_i = 1$, for all $i \in V$. We denote the set of feasible solutions of the above LP as $P_t(G)$, or much simpler as P_t when the underlying graph is clear from the context, and we call it the t-partial vertex-cover polytope. For each edge e, the reader should understand x_e as the 0-1 indicator variable that says whether e will be among the (at least) t many that will be covered by some vertex, while for each vertex i, the 0-1 variables x_i indicate whether vertex i is chosen in the solution.

(t-PVC-LP) is the starting point for the 2-approximation algorithm for t-PVC in [4], and a $2 - \Theta(1/d)$ approximation for unweighted instances, where d the maximum degree of the input graph, in [29, 11]. Strictly speaking, the analysis that guarantees the 2-approximability is not relative to the performance of the LP for all instances, as in fact (t-PVC-LP) has an unbounded integrality gap.

▶ **Observation 1** (Star-graph fools (t-PVC-LP) [25]). *Consider the unweighted star-graph $G = (V, E)$ with $V = 1, \dots, n, n+1$, and edges $\{n+1, i\} \in E$, for $i = 1, \dots, n$. The optimal solution to t-PVC is 1, for every $t \in \mathbb{N}$. In contrast, consider the feasible solution to (t-PVC-LP) that sets $x_e = x_{n+1} = t/n$ for all $e \in E$, and the rest of variables equal to 0. This gives a solution of cost t/n, hence the integrality gap of (t-PVC-LP) is at least n/t.*

This is also true for the SDP relaxation of t-PVC that gives the best known approximation algorithm.

▶ **Proposition 2.** *For all $t \le n/2$, the SDP of [15] has integrality gap at least n/t when the input is the star-graph of Observation 1.*

2.2 Hierarchies of LP and SDP relaxations

In this section we introduce families of LPs and SDPs derived by the so-called LS, LS$_+$ [23] and SA [28] systems. Starting with a polytope $P \subseteq [0,1]^m$, each of the LS$_+$ and SA systems derives a nested sequence of relaxations $\{P^{(r)}\}_{r=1,\dots,m}$, such that $P^{(m)} = conv\,(P \cap \{0,1\}^m)$, while under mild assumptions one can optimize over $P^{(r)}$ in time $m^{O(r)}$. For an instance $G = (V, E)$ of t-PVC, our intention is to derive and study this sequence of relaxations starting with $P = P_t(G)$, i.e. the feasible region of the standard LP relaxation (t-PVC-LP), hence setting $|m| = |V| + |E|$. For the sake of simplicity, we adopt a unified exposition of the systems (see [22] for a more abstract exposition of lift-and-project systems).

For technical reasons, it is convenient to apply a standard homogenization to polytope P as follows: variables x_p are replaced by $\overline{x}_{\{p\}}$ and each constraint $a^T x \ge b$ is replaced by $a^T \overline{x} \ge b \overline{x}_\emptyset$. Adding the constraint $\overline{x}_\emptyset \ge 0$ along with the previous constraints define a cone that we denote by K. Clearly $K \cap \{\overline{x}_\emptyset = 1\}$ is exactly polytope P. Next we define a sequence of SDP refinements of an arbitrary 0-1 polytope, proposed by Lovász and Schrijver [23], and that is commonly known in the literature as the LS$_+$-hierarchy (of SDPs).

▶ **Definition 3** (The LS$_+$ system). Let $K^{(0)} := K$ be a conified polytope $P \subseteq [0,1]^m$. The level-r LS$_+$tightening of $K^{(0)}$ is defined as the cone

$$K^{(r)} = \left\{ x \in \mathbb{R}^{\mathcal{P}_1} : \exists y \in \mathbb{R}^{\mathcal{P}_2} : \begin{array}{l} \mathcal{Y} \succeq \mathbf{0}, \ \ \mathcal{Y}\mathbf{e}_\emptyset = x \ \text{ and} \\ \forall i \in [m], \ \mathcal{Y}\mathbf{e}_{\{i\}}, \mathcal{Y}\left(\mathbf{e}_\emptyset - \mathbf{e}_{\{i\}}\right) \in K^{(r-1)} \end{array} \right\}$$

The level-r LS$_+$ refinements (tightenings) $\mathcal{N}_+^{(r)}(P)$ of P is obtained by projecting $K_+^{(r)}$ onto $x_\emptyset = 1$, i.e. $\mathcal{N}_+^{(r)}(P) = K_+^{(r)} \cap \{x \in \mathbb{R}^{\mathcal{P}_1} : \ x_\emptyset = 1\}$.

Next we introduce the SA system defined by Sherali and Adams [28], and that derives a sequence of LP relaxations (and not SDP relaxations).

▶ **Definition 4** (The SA system). Let K be a conified polytope $P \subseteq [0,1]^m$. The level-r SA tightening of K is defined as the cone

$$M^{(r)} = \left\{ x \in \mathbb{R}^{\mathcal{P}_1} : \ \exists y \in \mathbb{R}^{\mathcal{P}_{r+1}} : \begin{array}{l} \mathcal{Y}\mathbf{e}_\emptyset = x, \text{ and} \\ \forall Y, N \text{ with } Y \cup N \in \mathcal{P}_r, \ \mathcal{Y} \sum_{\emptyset \subseteq T \subseteq N} (-1)^{|T|} \mathbf{e}_{Y \cup T} \in K \end{array} \right\}$$

The level-r SA refinement (tightening) $\mathcal{S}^{(r)}(P)$ of P is obtained by projecting $M^{(r)}$ onto $x_\emptyset = 1$, i.e. $\mathcal{S}^{(r)}(P) = M^{(r)} \cap \{x \in \mathbb{R}^{\mathcal{P}_1} : \ x_\emptyset = 1\}$.

Occasionally we abuse notation and we treat $\mathcal{N}_+^{(r)}(P), \mathcal{S}^{(r)}(P)$ as subsets of $[0,1]^m$, instead of $\{x \in [0,1]^{m+1} : x_\emptyset = 1\}$. Also, relaxations derived by LS$_+$ and SA are in principle incomparable.

We observe that level-1 SA tightening coincides with the level-1 Lovász-Schrijver-LP tightening ($\mathcal{N}_+^{(1)}(P)$ without the PSD constraint). This seemingly weak LP solves the problematic star graph.

▶ **Proposition 5.** *Let G be the star graph of Observation 1. Then the level-1 SA tightening of $P_t(G)$ has integrality gap 1.*

Proof. Let x be a vector in the level-1 SA tightening of $P_t(G)$, and let y be its moment matrix $\mathcal{Y}$ as in Definition 4. Suppose now that for some $\mathbf{b}, \overline{\mathbf{b}}, \mathbf{d}, \overline{\mathbf{d}} \in \mathbb{R}^n$ and $a \in \mathbb{R}$ we have $\mathcal{Y}\mathbf{e}_\emptyset^T = \left(1, \mathbf{b}^T, a, \mathbf{d}^T\right)$ and $\mathcal{Y}\mathbf{e}_{\{n+1\}}^T = \left(a, \overline{\mathbf{b}}^T, a, \overline{\mathbf{d}}^T\right)$, where we explicitly assume that the list of indices has first all vertices (with the center being last), followed by all edges. Note that with this terminology, the value of the objective for such a solution is $a + \mathbf{1}_n^T\mathbf{b}$, which we need to compare to *opt* $= 1$.

Next we focus on $\mathcal{Y}(\mathbf{e}_\emptyset - \mathbf{e}_{\{n+1\}})$ that satisfies all homogenized constraints of $P_t(G)$, and in particular constraints (1) of edges $\{n+1, i\}$, $i = 1, \ldots, n$, which require that $\mathbf{b} - \overline{\mathbf{b}} \geq \mathbf{d} - \overline{\mathbf{d}}$. Similarly, constraint (2) of $P_t(G)$ implies that $\mathbf{1}_n^T(\mathbf{d} - \overline{\mathbf{d}}) \geq (1-a)t$. Therefore $a + \mathbf{1}_n^T\mathbf{b} \geq a + \mathbf{1}_n^T(\mathbf{d} - \overline{\mathbf{d}}) \geq a + (1-a)t \geq 1 = opt$. ◀

Recall that by Proposition 2 the star graph is also responsible for a n/t integrality gap for the SDP of [15], i.e. the relaxation which the best algorithm known for t-PVC is based on. The surprising conclusion from Proposition 5 is that a simple LP that one can derive systematically from $P_t(G)$ outperforms that particular SDP for a specific instance. This is in contrast to other known examples of level-$\Theta(m)$ LS tightenings that are strictly weaker than natural and static SDP relaxations. Finally, it is worthwhile mentioning that we do not know whether constant-level L&P tightenings of (t-PVC-LP) derive the SDP of [15].

For algorithmic purposes, a number of SA variants have been proposed that give rise to hierarchies of SDPs (see [1] for a list of them). The simplest variation, and the one that has resulted surprisingly strong positive results, is usually referred as the mixed hierarchy. This system, that we denote here by SA$_+$ imposes an additional PSD constraint.

▶ **Definition 6** (The SA$_+$ system). Let K be a conified polytope $P \subseteq [0,1]^m$. The level-r SA$_+$ tightening of K is defined as the refinement of cone $M^{(r)}$, as in Definition 4, where the $(m+1)$-leading principal minor of the moment matrix $\mathcal{Y}$, i.e. the principal minor of $\mathcal{Y}$ that is indexed by sets of variables of size at most 1, is PSD.

Level-r SDPs derived by the SA$_+$ and LS$_+$ systems are not comparable. In Section 4 we introduce a further refinement of SA$_+$ that is strictly tighter than LS$_+$, and for which we actually derive the same IG lower bounds as in SA. We postpone its definition due to its technicality.

By the generic algorithmic properties common to LS$_+$, SA and SA$_+$ systems, and for the t-PVC polytope, it is immediate that for any graph $G = (V, E)$ the level-($|V| + |E|$) relaxations have integrality gap 1. However, from the proof of convergence from all systems, it easily follows that vectors in level-r relaxations satisfy *any* constraint that is valid for the integral hull of $P_t(G)$ and that has support at most r. If *opt* denotes the optimal value for $G = (V, E)$ then $\sum_{i \in V} x_i \geq opt$ is a constraint valid for every integral solution with support $|V|$. Hence, level-$|V|$ LPs or SDPs derived by SA, LS$_+$ and SA$_+$ systems can solve any t-PVC instance exactly. Can level-r relaxations close the unbounded integrality gap of $P_t(G)$ as exhibited in Observation 1, for $r = o(|V|)$? We answer this question in the negative in the next sections by proving strong integrality gaps for superconstant level LP and SDP relaxations. As a byproduct, we show this way that LPs and SDPs that give rise to algorithms that run in superpolynomial time cannot solve to any good proximity even the tractable combinatorial problem t-PVC where $t = \Theta(1)$.

3 IG lower bounds for the Sherali-Adams LP system

This section is devoted in proving one of our main results.

▶ **Theorem 7.** *Let n, r, t be integers with $n \geq 2r + 2t + 2$. Then the integrality gap of the level-r SA-tightening of (t-PVC-LP) on graphs with n vertices is at least $\binom{n-2r}{2}/t \cdot n$.*

For this we fix a clique $G = (V, E)$ on n vertices, along with r, t such that $n \geq 2r + 2t + 2$. We start by presenting Random Process 1, that defines a distribution of 0-1 assignments for variables of the polytope $P_t(G)$.

Random Process 1 (Definition of distribution $\mathcal{D}_p$)

```
Require: A fixed $p \in [0, 1]$.
1: for $i \in V$ do
2:     Independently at random, set $x_i = 1$ with probability $p$
3: end for
4: for $e \in E$ do
5:     Set $x_e$ equal to 1 as long as $e$ is incident to some $i$ for which $x_i = 1$, and otherwise to 0.
6: end for
Output: Distribution $\mathcal{D}_p$ induced by the experiment above.
```

We are ready to propose a vector solution $y \in \mathbb{R}^{\mathcal{P}_{r+1}}$ to the level-r SA tightening of $P_t(G)$. For $A \in \mathcal{P}_{r+1}$ (with ground set $V \cup E$), and for each $q \in A$, let X_q be the random variable which equals 1 if $x_q = 1$ in the random experiment of $\mathcal{D}_p$, and 0 otherwise. For all such $A \subseteq V \cup E$, we define

$$y_A := \mathop{\mathbb{E}}_{\mathcal{D}_p}\left[\prod_{q \in A} X_q\right] = \mathop{\mathbb{P}}_{\mathcal{D}_p}\left[\forall q \in A,\ x_q = 1\right] \tag{4}$$

where the last equality is due to that X_q are 0-1 variables. In particular, this means that for all $i \in V$ and $f \in E$ we have $y_{\{i\}} = p, \quad y_{\{f\}} = 2p - p^2$.

We use the following technical lemma; it is a standard observation used in many SA lower bounds.

▶ **Lemma 8.** *For $Y \cup N \in \mathcal{P}_{r+1}$, let $w_{Y,N} := \sum_{\emptyset \subseteq T \subseteq N} (-1)^{|T|} y_{Y \cup T}$. Then $w_{Y,N} = \mathbb{P}_{\mathcal{D}_p(Y \cup N)}\left[\forall q \in Y, X_q = 1, \ \& \ \forall q' \in N, X_{q'} = 0\right]$.*

We can now prove that y is solution to the level-r SA polytope of t-PVC, for a proper choice of p.

▶ **Lemma 9.** *For the complete graph $G = (V, E)$ on n vertices, and for all r, t with $n \geq 2r + 2t + 2$, let $y \in \mathbb{R}^{\mathcal{P}_{r+1}}$ be as in (4), where $p = t/\binom{n-2r}{2}$. Then $y \in S^{(r)}(P_t(G))$.*

Proof. Let $Y, N \in \mathcal{P}_r$ with $|Y \cup N| \leq t$. We need to show that $\overline{y} := \mathcal{Y} \sum_{\emptyset \subseteq T \subseteq N} (-1)^{|T|} \mathbf{e}_{Y \cup T} \in \mathbb{R}^{\mathcal{P}_1}$ satisfies all constraints of $P_t(G)$ (after they are homogenized).

Asking that $\overline{y}$ satisfies the constraint (1) for an edge $e = \{i, j\}$ is the same as asking that $w_{Y\cup\{i\},N} + w_{Y\cup\{j\},N} - w_{Y\cup\{e\},N} \geq 0$. Note that $|Y \cup N \cup \{i, j\}| \leq r + 2$. Due to Lemma (8) and by linearity of expectation we have

$$w_{Y\cup\{i\},N} + w_{Y\cup\{j\},N} - w_{Y\cup\{e\},N} = \underset{\mathcal{D}_p(Y\cup N\cup\{i,j\})}{\mathbb{E}} \left[\prod_{q\in Y} X_q \prod_{p\in N} (1 - X_p)\,(X_i + X_j - X_e) \right].$$

Recall in Random Process 1 we set $x_e = 1$ only when at least one among x_i, x_j is already set to 1. Therefore the previous expected value is always non negative.

In a similar manner we can show that box constraints (3) are satisfied. First, constraints of the form $x_q \geq 0$, $q \in V \cup E$ are satisfied for $\overline{y}$, since by Lemma 8, $w_{Y\cup\{q\},N}$ represents a probability of an event. As for constraints $x_q \leq 1$, we need to prove that $w_{Y\cup\{q\},N} \leq w_{Y,N}$. This is true again due to Lemma 8, and because the event associated with $w_{Y,N}$ is logically implied by that of $w_{Y\cup\{q\},N}$.

Finally we need to show that $\overline{y}$ satisfies constraint (2), i.e. constraint $\sum_{e\in E} w_{Y\cup\{e\},N} \geq t \cdot w_{Y,N}$. For this we recall that $|Y \cup N| \leq r$, and so in the original clique on n vertices, there is a subclique $G' = (U, F)$ on at least $n - 2r \geq 4$ vertices, such that no edge in F is incident to any element (vertex or edge) in $Y \cup N$, and $|F| \geq \binom{n-2r}{2} > 0$. This means that for every $f \in F$ the event that $X_f = 1$ is independent to any 0-1 assignment on variables in $Y \cup N$, while $\mathbb{P}_{\mathcal{D}_p}[X_f = 1] = 2p - p^2 \geq p$, since $p = t/\binom{n-2r}{2} \leq t/\binom{2t+2}{2} < 1/2$. Since we also have $|F| \cdot p = |F| \cdot t/\binom{n-2r}{2} \geq t$, we conclude that $\sum_{e\in E} w_{Y\cup\{e\},N} \geq \sum_{e\in F} w_{Y\cup\{e\},N} = |F| \cdot p \cdot w_{Y,N} \geq t \cdot w_{Y,N}$, as promised. ◀

The objective of the level-r SA LP is no more than $n \cdot p = t \cdot n/\binom{n-2r}{2}$, while the optimal solution of the input graph has cost 1, concluding the proof of Theorem 7.

It is worthwhile noticing that our superconstant integrality gaps lower bounds hold only for values of parameter $t = o(n)$. The reader can easily verify that when the input is the n-clique, then the optimal solution to (t-PVC-LP) is exactly $t/(n-1)$ (e.g. using the dual of (t-PVC-LP)). Therefore, for any constant c and when $n/c \leq t \leq n - 1$, for which the optimal solution to t-PVC is still 1, the integrality gap of (t-PVC-LP) is strictly less than c.

4 IG lower bounds for various SDP hierarchies

4.1 SDPs derived by the SA$_+$ and LS$_+$ systems

In this section we argue that the moment matrix $\mathcal{Y}$ of solution y that we proposed in Lemma 9 satisfies very strong PSD conditions. This will immediately imply the same IG lower bounds of Theorem 7 also for stronger SDP systems, as summarized in the next theorem.

▶ **Theorem 10.** *Let n, r, t be integers with $n \geq 2r + 2t + 2$. Then the integrality gap of the level-r LS$_+$ and SA$_+$ tightenings of* (t-PVC-LP) *on graphs with n vertices is at least $\binom{n-2r}{2}/t \cdot n$.*

For proving Theorem 10, we fix the clique $G = (V, E)$ on n vertices, together with r, t such that $n \geq 2r + 2t + 2$. In all our arguments below we use $y \in \mathbb{R}^{\mathcal{P}_{r+1}}$ as defined in (4), as well as vector w (indexed by pairs of sets of variables) as it appears in Lemma 8. We also define the matrix $\mathcal{X}^{Y,N} \in \mathbb{R}^{\mathcal{P}_1 \times \mathcal{P}_1}$, which at entry A, B (i.e. any two sets of size at most 1) equals $w_{Y\cup A\cup B,N}$. Note that matrix $\mathcal{X}^{Y,N}$ is exactly the moment matrix of random variables $\{X_q\}_{q\in V\cup E}$ condition on $X_q = 1$ for all $q \in Y$, and $X_{q'} = 0$ for all $q' \in N$, scaled by the constant $\mathbb{P}_{\mathcal{D}_p}[\forall q \in Y, X_q = 1 \;\&\; \forall q' \in N, X_{q'} = 0]$. In particular, for each $q \in V \cup E$ we have that vectors $\mathcal{X}^{Y,N}\mathbf{e}_q, \mathcal{X}^{Y,N}(\mathbf{e}_\emptyset - \mathbf{e}_q)$ satisfy all constraints of $P_t(G)$.

Now recall that $y \in \mathbb{R}^{\mathcal{P}_{r+1}}$ is obtained by the global distribution $\mathcal{D}_p$ that associates any 0-1 assignment of variables of $P_t(G)$ with some probability. In particular, if $x \in \{0,1\}^{\mathcal{P}_1}$, with $x_\emptyset = 1$, is such a 0-1 assignment, then xx^T is a rank 1 PSD matrix. Clearly, matrix $\mathcal{X}^{Y,N}$ is a convex combination of such rank-1 PSD matrices, hence it is PSD as well. We conclude with an Observation.

▶ **Observation 11.** *Let Y, N be any subsets of $V \cup E$ such that $|Y \cup N| \le r-1$. Then $\mathcal{X}^{Y,N}$ is positive semidefinite.*

It is now immediate that our SA solution y satisfies also the extra PSD constraint imposed by SA$_+$. What we only need to observe is that the leading principal minor of $\mathcal{Y}$ indexed by sets of size at most 1 is exactly $\mathcal{X}^{\emptyset,\emptyset}$, which is PSD by Observation 11. Hence, Theorem 7 also holds when SA tightenings are replaced by SA$_+$ tightenings.

Next we argue that our SA solution is robust against much stronger SDP refinements. Note that vector w is well defined for all level-r SA solutions y. Especially when y is obtained as a convex combination of integral vectors, all matrices $\mathcal{X}^{Y,N}$ are PSD, for all $|Y \cup N| \le r-1$. That is, the latter constraints constitute a further refinement of the SA$_+$ system. Again by Observation 11 it is immediate that our level-r SA solution fools also these exponentially many (in r) PSD conditions. These new PSD refinements are stronger than the constraints derived by the level-$(r-1)$ LS$_+$ system (see [30]); this concludes the proof of Theorem 10.

4.2 On SDPs derived by the Lasserre system

A natural question to ask is whether our SA solution fools SDPs derived by the so-called Lasserre (La) system [21]. The level-r La-SDP is defined as follows. For $y \in \mathbb{R}^{2r+2}$, the La-moment matrix $\mathcal{Z}$ is a matrix indexed by $\mathcal{P}_{r+1}$ with $\mathcal{Z}_{A,B} = y_{A\cup B}$. For each constraint $\sum_i \alpha_i^{(l)} x_i - \beta^{(l)} \ge 0$ of P, the La-slack moment matrix $\mathcal{Z}^{(l)}$ is a matrix indexed by $\mathcal{P}_r$ with $\mathcal{Z}^{(l)}_{A,B} = \sum_i \alpha_i^{(l)} y_{A\cup B\cup\{i\}} - \beta^{(l)} y_{A\cup B}$. The level-$r$ La SDP requires that all matrices $\mathcal{Z}$ and $\{\mathcal{Z}^{(l)}\}_l$ are PSD. Notably, the PSDness of proper principal minors of matrices $\mathcal{Z}$ and $\{\mathcal{Z}^{(l)}\}_l$ imply the level-r SA linear constraints [22]. As such, the level-r La SDP is at least as strong as the level-r SA LP. Unfortunately, we show that the level-1 La SDP is not fooled by our SA solution.

▶ **Lemma 12.** *For any constant r, the level-1 La SDP eliminates the level-r solution proposed in Lemma 9.*

Proof. Fix n, t, p, and let y be the solution to the level-(r) SA-tightening as described in Lemma 9. Consider the level-1 slack matrix for (2). In order to prove that this matrix is not PSD, it suffices to focus on its principal minor $\overline{\mathcal{Z}}$ that is indexed only by subsets of vertices. To that end, let $y_A \in \mathbb{R}^{\mathcal{P}_1}$ be the indicator vector of set $A \subseteq V$. Let also S_n denote the expected slack we have in constraint (2) when each vertex is chosen with probability p in the n-clique, and $C_{n,a}$ be the number of edges that are covered by choosing a many vertices in the same graph. Then, it is easy to verify by definition that $\overline{\mathcal{Z}}$ has the form $\left(\overline{\mathcal{Z}}\right)_{I,J} = p^{|I\cup J|}\left(S_{n-|I\cup J|} + C_{n,|I\cup J|}\right)$. Applying the Schur complement on $\overline{\mathcal{Z}}$ with respect to the entry $\left(\overline{\mathcal{Z}}\right)_{\emptyset,\emptyset} = S_n$, and given that $S_n > 0$, we have that $\overline{\mathcal{Z}}$ is PSD if and only if $M - \frac{(p(S_{n-1}+C_{n,1}))^2}{S_n} J_n$ is PSD, where M is the minor of $\overline{\mathcal{Z}}$ indexed by sets of vertices of size 1, and J_n is the all-one $n \times n$ matrix. By symmetry, all rows of M have the same sum, i.e. the all-one vector $\mathbf{1}$ is an eigenvector for the Schur complement. Elementary calculations then show that the leading term of the corresponding eigenvalue, when $p = c/n^2$, is $\left(-2c^4 - \frac{15c^3}{2} - 2c^2\right)\frac{1}{n} < 0$ (the rest of the summands are of order $o(1/n)$). ◀

5 Discussion / Open Problems

The algorithmic significance of our results pose a natural (and classic) open problem, related also to questions on extended formulations; Does t-PVC admit a polysize (or tractable) LP or SDP relaxation that has integrality gap no more than 2, even when $t = O(n)$? It is notable that this question has been studied in [3] for a generalization of t-PVC but with no implications to our problem. Note also that our strongest IG lower bounds are valid only when $t/n = \epsilon$, for small enough $\epsilon > 0$, where n is the number of vertices of the input graph. As a result, another interesting open question is, given t and n, find the smallest $r = r(n, t)$ for which the level-r LP or SDP derived by some L&P system has integrality gap no more than 2. In particular, can it be that $r = \omega(1)$ when $t \geq n$?

Finally, our SDP IG lower bounds make explicit that global distributions of 0-1 assignments can be used to witness solutions to SA LP tightenings of superconstant integrality gaps. We also demonstrate that it is almost straightforward to show that the same solutions are robust against SDP tightenings of many L&P systems except the La system. Can the same family of global distributions fool La SDPs when it is also enriched with intuitive and stronger conditions? A generic positive or negative answer would give new insights in understanding the power of the various SDP hierarchies.

Acknowledgments. We would like to thank the anonymous referees for their valuable comments.

References

1 Y. Au and L. Tunçel. A comprehensive analysis of polyhedral lift-and-project methods. *CoRR*, abs/1312.5972, 2013.

2 S. Benabbas, S. O. Chan, K. Georgiou, and A. Magen. Tight Gaps for Vertex Cover in the Sherali-Adams SDP Hierarchy. In *FSTTCS*, volume 13 of *LIPIcs*, pages 41–54, 2011.

3 S. K. Bera, S. Gupta, A. Kumar, and S. Roy. Approximation algorithms for the partition vertex cover problem. In *WALCOM*, volume 7748 of *LNCS*, pages 137–145. Springer, 2013.

4 N. H. Bshouty and L. Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *STACS*, volume 1373 of *LNCS*, pages 298–308. Springer, 1998.

5 B. Caskurlu and K. Subramani. On partial vertex cover on bipartite graphs and trees. *CoRR*, abs/1304.5934, 2013.

6 M. Charikar, K. Makarychev, and Y. Makarychev. Integrality gaps for Sherali-Adams relaxations. In *STOC*, pages 283–292, New York, NY, USA, 2009. ACM Press.

7 K. K. H. Cheung. Computation of the lasserre ranks of some polytopes. *Math. Oper. Res*, 32(1), 2007.

8 E. Chlamtáč and M. Tulsiani. Convex relaxations and integrality gaps. In Miguel F. Anjos and Jean B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*, volume 166 of *International Series in Operations Research & Management Science*, pages 139–169. Springer US, 2012.

9 W. Cook and S. Dash. On the matrix-cut rank of polyhedra. *Mathematics of Operations Research*, 26(1):19–30, 2001.

10 Irit Dinur and Shmuel Safra. On the hardness of approximating minimum vertex-cover. *Annals of Mathematics*, 162(1):439–486, 2005.

11 R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

12 K. Georgiou and E. Lee. Lift and project systems performing on the partial-vertex-cover polytope. *CoRR*, abs/1409.6365v1, 2014.

13 K. Georgiou and A. Magen. Expansion Fools the Sherali-Adams System: Compromising Local and Global Arguments. Technical Report CSRG-587, University of Toronto, November 2008.

14 K. Georgiou, A. Magen, T. Pitassi, and I. Tourlakis. Integrality gaps of 2-o(1) for vertex cover SDPs in the Lovász–Schrijver hierarchy. *SIAM J. Comput*, 39(8):3553–3570, 2010.

15 E. Halperin and A. Srinivasan. Improved approximation algorithms for the partial vertex cover problem. In *APPROX*, volume 2462, pages 161–174. Springer, 2002.

16 D. S. Hochbaum. The t-vertex cover problem: Extending the half integrality framework with budget constraints. In *APPROX*, volume 1444 of *LNCS*, pages 111–122. Springer Berlin Heidelberg, 1998.

17 C. Kenyon-Mathieu and W. F. de la Vega. Linear programming relaxations of maxcut. In *SODA*, pages 53–61. ACM Press, 2007.

18 S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-ϵ. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

19 S. G. Kolliopoulos and Y. Moysoglou. Sherali-Adams gaps, flow-cover inequalities and generalized configurations for capacity-constrained Facility Location. In *APPROX*, LNCS, page to appear, 2014.

20 J. Könemann, O. Parekh, and D. Segev. A unified approach to approximating partial covering problems. *Algorithmica*, 59:489–509, 2011.

21 J. B. Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. In *IPCO*, volume 2081 of *LNCS*, pages 293–303. Springer, Berlin, 2001.

22 M. Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 programming. *Math. Oper. Res.*, 28(3):470–496, 2003.

23 L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, May 1991.

24 M. Mastrolilli. The lasserre hierarchy in almost diagonal form. *CoRR*, abs/1312.6493, 2013.

25 J. Mestre. A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. *Algorithmica*, 55(1):227–239, 2009.

26 G. Schoenebeck. Linear level lasserre lower bounds for certain k-CSPs. In *FOCS*, pages 593–602. IEEE Computer Society, 2008.

27 G. Schoenebeck, L. Trevisan, and M. Tulsiani. Tight integrality gaps for Lovász-Schrijver LP relaxations of vertex cover and max cut. In *STOC*, pages 302–310. ACM Press, 2007.

28 H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for 0–1 programming problems. *SIAM J. Discrete Math.*, 3(3):411–430, 1990.

29 A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *FOCS*, pages 588–599, 2001.

30 Iannis Tourlakis. *New lower bounds for approximation algorithms in the Lóvasz-Schrijver hierarchy.* PhD thesis, Department of Computer Science, June 2006.

31 J. Tu, J. Du, and F. Yang. An iterative rounding 2-approximation algorithm for the k-partial vertex cover problem. *Acta Mathematicae Applicatae Sinica, English Series*, 30(2):271–278, 2014.

32 M. Tulsiani. CSP gaps and reductions in the Lasserre hierarchy. In *STOC*, pages 303–312, New York, NY, USA, 2009. ACM Press.

Replica Placement on Directed Acyclic Graphs

Sonika Arora[1], Venkatesan T. Chakaravarthy[2], Kanika Gupta[1], Neelima Gupta[1], and Yogish Sabharwal[2]

1 Department of Computer Science, University of Delhi, India
sonika.ta@gmail.com,kanika.g.mcs.du.2012@gmail.com,ngupta@cs.du.ac.in

2 IBM India Research Lab, New Delhi, India
{vechakra,ysabharwal}@in.ibm.com

Abstract

The replica placement problem has been well studied on trees. In this paper, we study this problem on directed acyclic graphs. The replica placement problem on general DAGs generalizes the set cover problem. We present a constant factor approximation algorithm for the special case of DAGs having bounded degree and bounded tree-width (BDBT-DAGs). We also present a constant factor approximation algorithm for DAGs composed of local BDBT-DAGs connected in a tree like manner (TBDBT-DAGs). The latter class of DAGs generalizes trees as well; we improve upon the previously best known approximation ratio for the problem on trees. Our algorithms are based on the LP rounding technique; the core component of our algorithm exploits the structural properties of tree-decompositions to massage the LP solution into an integral solution.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, LP Rounding

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.213

1 Introduction

The replica placement problem is an important problem that finds applications in a variety of domains such as internet and video on demand service delivery (see [8, 10, 5]). We refer to [13] for additional applications. This problem is concerned with the optimal placement of copies (replicas) of a database on the nodes of a network in order to serve periodic requests from a set of clients under the setting wherein each replica can serve a limited number of requests and a client can only be served by a replica within a specified distance (QOS requirement). Prior work has studied the problem for the case of tree networks [5, 13, 3, 9, 2, 1]. The goal of this paper is to address the problem on more general DAG networks.

Replica Placement Problem. The input consists of a DAG $G = (V, E)$. Each leaf node (having no in-edges) represents a *client*. Let A be the set of all the clients and let $|A| = m$. The input specifies a *request* $r(a)$ for each client $a \in A$. The input also includes a *capacity* W. For each edge (u, v) in E, the input specifies a distance $d(u, v)$. For a node u and a client a such that there is a path from a to u in the graph, let $d(a, u)$ be the shortest distance from a to u. Each client a is associated with a quantity $d_{\max}(a)$, the maximum distance it can travel.

A feasible solution selects a subset of nodes and places replicas on them in order to service the requests of the clients. The solution must assign the request of each client a to some (unique) replica u such that there is a path from a to u in the graph and $d(a, u) \leq d_{\max}(a)$; we call this latter condition the *distance constraint*. Furthermore, the total requests assigned to any replica must not exceed the capacity W. A client can be serviced by opening a replica at the client node itself. Our goal is to minimize the number of replicas placed.

We assume that the capacity W and the requests $r(\cdot)$ are integral and that W is polynomially bounded in the number of nodes. Furthermore, without loss of generality, we assume that $r(a) \leq W$ for all clients $a \in A$.

Prior Work. The above problem and its variants have been well-studied for tree networks in the existing literature [5, 13, 3, 9, 2, 1], from both practical and algorithmic perspectives.

Benoit et al. [3] studied the the replica placement problem on trees. They showed that the problem is NP-hard to approximate within a factor of 3/2 even without distance constraints (i. e., $d_{\max}(a) = \infty$, for all clients a) and when the network is a binary tree. They obtained the above result by showing that the above problem generalizes the bin-packing problem. Benoit et al. [2] presented a 2-approximation for the replica placement problem without distances. For the case with distances, they designed a greedy algorithm with an approximation ratio of $(1 + \Delta)$, where Δ is the maximum number of children of any node.

Arora et al.[1] presented an algorithm for tree networks with a constant approximation ratio (independent of Δ). Their result also applies to the partial cover version, wherein the input additionally specifies a number K and only K clients need to be serviced by a solution.

The replica placement problem can easily be seen to be a special case of the capacitated set cover problem. The latter problem admits an algorithm with an approximation ratio of $O(\log n)$ [12]; up to constant factors, the ratio is the best possible, unless NP = P [6]. For the case of vertex cover, Chuzhoy and Naor [4] and Gandhi et al. [7] presented algorithms for the capacitated vertex cover problem with approximation ratio of 3 and 2, respectively. However, their algorithms can handle only the case of simple graphs. Saha and Khuller [11] presented a 34-approximation algorithm for the more general case of multi-graphs.

The capacitated vertex cover problem plays an important role in the constant factor approximation algorithm for the replica placement problem on tree networks, due to Arora et al. [1], mentioned earlier. The above algorithm is based on the LP rounding technique. It works by reducing the issue of rounding an LP solution for the replica placement problem to an issue of rounding LP solutions of a suitably construed capacitated vertex cover instance. The algorithms presented in the current paper also make use of the above strategy.

Our Results. Prior work has primarily dealt with the replica placement problem on tree networks. In this paper, we study the problem on directed acyclic graphs. On DAGs, the replica placement problem is as hard as the capacitated set cover problem. We therefore focus on special classes of DAGs and provide constant factor approximation algorithms.

The first class of DAGs that we address are rooted DAGs that have bounded degree and bounded tree-width; we call these BDBT-DAGs. Tree-width is a notion traditionally associated with undirected graphs. A graph with bounded tree-width can be decomposed into disconnected pieces by removing a small number of nodes. By tree-width of a DAG, we shall mean the tree-width of the graph obtained by ignoring the direction on the edges.

Our first result is a constant factor approximation algorithm for BDBT-DAGs.

- There exists a polynomial time algorithm for the replica placement problem on BDBT-DAGs having an approximation ratio of $2 \cdot (d + t + 2)$, where d and t are respectively the degree bound and tree-width bound. of the input BDBT-DAG.

Our second result deals with a generalization of BDBT-DAGs, which we call TBDBT-DAGs (tree of BDBT-DAGs). Intuitively, a TBDBT-DAG is composed of BDBT-DAGs connected in a tree-like manner. A TBDBT-DAG is constructed by starting with a skeletal

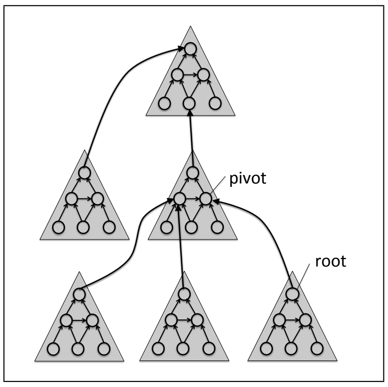


Figure 1 Example of TBDBT-DAG. The figure also shows the root of one of the component BDBT-DAGs and the pivot that it connects to in another BDBT-DAG.

tree $\mathcal{T}$, whose vertices are referred to as proxies. Then, each proxy is substituted with a BDBT-DAG. For each proxy q in $\mathcal{T}$, the root vertex of the associated BDBT-DAG is connected to some vertex (called pivot) in the BDBT-DAG associated with the parent proxy of q (see Figure 1).

The overall DAG has bounded tree-width, but may not have bounded degree. Our main result is a constant factor approximation algorithm for TBDBT-DAGs.

- There exists a polynomial time algorithm for the replica placement problem on TBDBT-DAGs having an approximation ratio of $O(d + t)$, where d and t are respectively the maximum degree bound and maximum tree-width bound of any component BDBT-DAG of the input TBDBT-DAG.

The class of TBDBT-DAGs clearly generalizes trees (wherein each component BDBT-DAG consists of a single vertex). Therefore the above result generalizes the constant factor approximation algorithm for the case of trees given in the prior work [1]. In fact, our analysis is more refined and leads to an improved constant factor.

Discussion. Tree networks, considered in prior work, have a simpler structure wherein each node has only one out-neighbor (i. e., parent). If we ignore the distance constraints, replica placement on such networks reduces to a capacitated set cover scenario over a set system consisting of a laminar family of sets. The classical set cover problem on such laminar familes can easily be handled. Thus, the distance and the capacity constraints are the only critical issue in the case of tree networks. On the the other hand, networks considered by us are DAGs, wherein a node can have multiple (but, bounded) number of out-neighbors leading to more complex set systems. In fact, any arbitrary set system can be encoded as an instance of the replica placement problem if we allow general DAGs, or even bounded degree DAGs. One of the important technical contributions of this paper is to show that the issue can be addressed, if the DAGs additionally have bounded tree-width (namely, BDBT-DAGs). However, the family of BDBT-DAGs do not encompass trees. Our main result deals with the larger class of TBDBT-DAGs, which generalizes both BDBT-DAGs and trees.

2 Preliminaries

In this section we describe a natural LP for our problem and setup terminology that we will use. We then formally define the specific class of DAGs that we address in this paper.

LP Formulation. We consider a natural LP formulation. We say that a solution *opens* node u, if it places a replica on it. We say that a client a is *attachable* to a node u, if there exists a path from a to u and $d(a,u) \le d_{\max}(a)$. For a client a, let $\mathtt{Att}(a)$ denote the set of all nodes to which client a can be attached. For a node u, let $\mathtt{Att}(u)$ denote the set of all clients that can be attached to node u. For a set of nodes U, let $\mathtt{Att}(U) = \cup_{u\in U}\mathtt{Att}(u)$.

For each node $u \in V$, we introduce a variable $y(u)$ that specifies the extent to which u is open. For each client $a \in A$ and each node $u \in \mathtt{Att}(a)$, we introduce a variable $x(a,u)$ that specifies the extent to which a is assigned to u. A node u is said to *service* a client a, if $x(a,u) > 0$; we also say that a is *assigned* to u if u services a.

$$\min \sum_{u\in V} y(u)$$

$$\sum_{a\in\mathtt{Att}(u)} x(a,u)\cdot r(a) \le y(u)\cdot W \qquad (\forall u \in V) \tag{1}$$

$$x(a,u) \le y(u) \qquad (\forall a \in A, u \in \mathtt{Att}(a)) \tag{2}$$

$$\sum_{u\in\mathtt{Att}(a)} x(a,u) = 1 \qquad (\forall a \in A) \tag{3}$$

$$y(u) \le 1 \qquad (\forall u \in V) \tag{4}$$

Further, we add non-negativity constraints for all the variables. Constraint (1) (called the *capacity constraint*) enforces that at any node the total request assigned does not exceed the capacity W. Constraint (2) ensures that a client can be assigned to a node only to an extent to which the node is open; without this constraint, it can be shown that the LP has an unbounded integrality gap. Constraint (3) enforces that every client is serviced to an extent of exactly one, i. e. every client is fully served. Constraint (4) requires that a node can be opened to an extent of at most one.

For an LP solution σ, we shall denote the variables of the solution by x_σ and y_σ unless stated otherwise. We shall also use x and y to denote the variables when the solution is clear from the context of the discussion. Consider an LP solution σ. The cost of an LP solution is given by the objective function: $\mathtt{cost}(\sigma) = \sum_{u\in V} y_\sigma(u)$. For a set $X \subseteq V$, the cost of the set X in the solution σ is given by $\mathtt{cost}_\sigma(X) = \sum_{u\in X} y_\sigma(u)$.

▶ **Definition 1** (Fully-open, fully-closed and partially-open nodes). We call a node u *fully-open*, if $y_\sigma(u) = 1$; *fully-closed*, if $y_\sigma(u) = 0$; and *partially open*, if $0 < y_\sigma(u) < 1$. ◀

▶ **Definition 2** (Load and Fully Loaded nodes). For a set of clients $B \subseteq A$ and set of nodes $U \subseteq V$, by $load_\sigma(B,U)$ we mean the total assignments of B to nodes in U, i. e., $load_\sigma(B,U) = \sum_{a\in B}\sum_{v\in U} x_\sigma(a,v)\cdot r(a)$.

For a node $u \in V$, By $load_\sigma(u)$ we mean the total assignments to node u, i. e., $load_\sigma(u) = \sum_{a\in\mathtt{Att}(u)} x_\sigma(a,u)\cdot r(a)$. A node u is said to *fully loaded* if $load_\sigma(u) = W$. ◀

▶ **Definition 3** (Integrally Open and Integral Solutions). A solution σ is said to be *integrally open*, if every node is either fully-open or fully-closed and an integrally open solution is said to be an *integral solution* if every client is serviced by exactly one node. ◀

DAGs of interest. In this paper we shall address two types of DAGs; these DAGs have bounded tree-width. We recollect the concept of tree-width and then formally define the DAGs that we address in this paper.

▶ **Definition 4** (Tree-decomposition and tree-width). A *tree-decomposition* of graph $G = (V,E)$ is a pair $\langle\{V_i | i \in I\}, T\rangle$ where $V_1, \dots, V_h$ are subsets of V called pieces, $I = [1,h]$, and T is a tree with elements of I as nodes. A tree-decomposition must satisfy the following properties:

1. *Node coverage*: every node of G belongs to at least one piece V_i, i. e., $\cup_{i \in I} V_i = V$
2. *Edge coverage*: for every edge $e = (u, v) \in E$, there is some piece V_i containing both ends of e, i. e., $\exists i \in I$ such that $\{u, v\} \subseteq V_i$
3. *Coherence*: for every graph vertex v , all pieces containing v form a connected component, i. e.,$\forall i, j, k \in I$, if j lies on the path between i and k in T, then $V_i \cap V_k \subseteq V_j$

The width of $\langle\{V_i | i \in I\}, T\rangle$ equals $max\{|V_i| : i \in I\} - 1$. The *tree-width* of G is the minimum k such that G has a tree-decomposition of width k. ◀

We will use the following property of tree-decompositions for designing our algorithm.

▶ **Property 1** (Separation Property). *Let p be any piece of T. Suppose that $T - p$ has components $T_1, T_2 .....T_d$. Then the subgraphs $G[T_1 - V_p], G[T_2 - V_p], \dots, G[T_d - V_p]$ have no nodes in common, and there are no edges between them.*

The first type of DAGs that we address have bounded degree and tree-width.

▶ **Definition 5** (Bounded Degree Bounded Tree-width DAG *(BDBT-DAG)*)**.** We say that a rooted DAG is a *bounded degree bounded tree-width DAG* if the undirected graph obtained by ignoring directions on the edges has bounded degree and bounded tree-width.

We denote the root node of a BDBT-DAG, G, by $\mathtt{Rt(G)}$. ◀

The second type of DAGs that we consider generalize BDBT-DAGs as well as trees.

▶ **Definition 6** (Tree of BDBT-DAGs *(TBDBT-DAG)*)**.** A TBDBT-DAG G is a pair $\langle\{D_j | j \in J\}, \mathcal{T}\rangle$ where $D_1, D_2, \dots, D_h$ are BDBT-DAGs, $J = [1, h]$ and $\mathcal{T}$ is a tree with the elements of J as nodes and labeled edges satisfying the following properties:

- The vertices of the BDBT-DAGs are disjoint, i. e., $V(D_i) \cap V(D_j) = \phi \; \forall \; i, j \in J, \; i \neq j$.
- The vertex set of G is the union of the vertices of BDBT-DAGs, i. e., $V(G) = \cup_{j \in J} V(D_j)$.
- The edges of all the BDBT-DAGs are contained in G, i. e., $E(D_j) \subseteq E(G)$ for all $j \in J$.
- For every edge $e = (D_i, D_j)$ in $\mathcal{T}$, there is an edge $(\mathtt{Rt(D_i)}, \ell(\mathtt{e}))$ in G that links the two BDBT-DAGs by connecting the root of D_i to a node of $V(D_j)$ determined by the label $\ell(e)$ on the edge; $\ell(e) \in V(D_j)$ is said to be a *pivot node*.

Let $\mathtt{Roots(G)}$ be the roots of all BDBT-DAGs of G, i. e., $\mathtt{Roots(G)} = \{\mathtt{Rt(D_j)} : \mathtt{j} \in \mathtt{J}\}$. ◀

We make the following observation regarding the tree-width of TBDBT-DAGs.

▶ **Observation 1.** *If the maximum tree-width of any component BDBT-DAG is t, then the TBDBT-DAG has tree-width* $\max\{t, 1\}$.

3 Algorithm for BDBT-DAGs via Stable Solutions

For the ease of exposition, we first consider the simpler case of BDBT-DAGs and present a constant factor approximation algorithm. The components developed as part of the algorithm will also be useful in handling the more generic TBDBT-DAGs.

The algorithm is based on the LP rounding technique. Let σ_{in} be an optimal solution to the LP formulation. The algorithm works by applying a sequence of transformations until an integral solution is obtained, wherein each transformation increases the cost by at most a constant factor. The notion of *stable solutions* plays a key role in the above process.

▶ **Definition 7** (Stable solution)**.** A solution σ is said to be *stable* if the nodes can be partitioned into two sets R and P called *rich* and *poor* nodes respectively such that the following properties are satisfied:

1. The rich nodes, R, are fully open.
2. For any poor node $u \in P$, the total extent to which the poor nodes service the clients attachable to u is less than W, i. e., $load_\sigma(\mathtt{Att}(u), P) < W$.
3. Every client is either serviced by only nodes in R or only nodes in P but not both. ◀

Intuitively, a stable solution segregates the input instance into two parts, the first part comprising of the rich nodes and the clients serviced by them, and the second part comprising of the poor nodes and the clients serviced by them. It is easy to handle the first part, since all the nodes in the instance are fully open. The second part has the useful feature that it is uncapacitated in essence; meaning, no matter how we assign the clients, the capacity W at a node can never be exceeded and hence, the capacity constraints can safely be ignored.

We next present two procedures. The first procedure works on any bounded degree DAG and transforms an arbitrary LP solution σ_{in} into a stable solution σ_s with only a $(d+2)$ factor increase in cost, where d is the degree bound. The second procedure works on any bounded tree-width DAG and transforms any stable solution σ_s into an integrally open solution σ_{io} with only a $(t+1)$ factor increase in cost, where t is the tree-width bound. Combining the two procedures, we can handle any BDBT-DAG and transform an arbitrary LP solution σ_{in} into an integrally open solution σ_{io} with only a constant factor increase in cost. Finally, the integrally open solution can be transformed into an integral solution using a cycle cancellation based method proposed in prior work [1].

The above procedures make use of a subroutine called *pulling procedure*, described next. This subroutine is also employed by other algorithms in the paper to reassign the clients.

Pulling Procedure. Given a node u and a set of nodes X such that $u \notin X$, by performing the *pulling* procedure from X on to u, we mean reassigning the clients that are attachable to u from X on to u while it has remaining capacity.

Formally, let σ_{in} be the input solution. We process each client-node pair $\langle a, v\rangle$ iteratively where $a \in Att(u)$ and $v \in X$. Let π_{old} be the LP solution at the start of the current iteration. We construct a new solution π_{new} as follows. We set $x_{\pi_{new}}(a,u) = x_{\pi_{old}}(a,u) + \delta$ and $x_{\pi_{new}}(a,v) = x_{\pi_{old}}(a,v) - \delta$ where

$$\delta = \min\left\{ x_{\pi_{old}}(a,v), \frac{W - load_{\pi_{old}}(u)}{r(a)} \right\}.$$

All other values of $x_{\pi_{new}}(.,.)$ and $y_{\pi_{new}}(.)$ are retained as in π_{old}. π_{new} is taken as the input solution π_{old} for the next iteration. At the end of processing all client-node pairs, the solution π_{new} of the last iteration is taken as the final solution σ_{out} output by this procedure.

Note that if $load_{\sigma_{in}}(u) + load_{\sigma_{in}}(\mathtt{Att}(u), X) \le W$ (before the pulling procedure), then $load_{\sigma_{out}}(\mathtt{Att}(u), X) = 0$, otherwise $load_{\sigma_{out}}(u) = W$ (after the pulling procedure).

3.1 Constructing Stable Solutions for Bounded Degree DAGs

In this section, we show how to transform any feasible solution σ_{in} for a bounded degree DAG into a stable solution σ_s as stated in the following Lemma.

▶ **Lemma 8.** *Any LP solution σ_{in} for a bounded degree DAG can be converted into a stable solution σ_s such that $\mathtt{cost}_{\sigma_s}(R) \le (d+1) \cdot \mathtt{cost}(\sigma_{in})$ and $\mathtt{cost}_{\sigma_s}(P) \le \mathtt{cost}(\sigma_{in})$ where R and P are respectively the rich and poor nodes of the stable solution σ_s.*

Proof Sketch. The transformation is divided into the *reddening phase* and *browning phase.*

Reddening Phase. In this phase, the algorithm tries to make as many nodes fully loaded as possible. We shall color all fully loaded nodes red. For this, the algorithm processes the nodes iteratively in an arbitrary order. For every node, it checks if the node can become fully loaded by performing the pulling procedure from non-red nodes. If so, we actually perform the pulling procedure, open this node and color it red. The reddening phase completes when all the nodes are processed. Hereafter, no node becomes fully loaded (red).

Browning Phase. In this phase, we process all the neighbors of red nodes that are not already red. For each such node, u, we open the node, perform the *pulling procedure* on u from non-red nodes and color it brown. Note that the total load on u, even after the reassignments, will be less than W (otherwise u would have become red in the first step). This completes the processing of the browning phase (and stage 1).

It can be shown that the solution at this stage is a stable solution taking the set of red and brown nodes to be rich and the remaining nodes to be poor. The first two properties are easy to check. The third property follows from the fact that if a client is attachable to a rich node and is assigned to a poor node, then it must be attachable to a brown node; but then the brown node would have pulled the assignments of this client from the nodes in P.

We now analyse the cost. Note that the cost of any feasible LP solution must be at least $\lfloor \sum_{a \in A} r(a)/W \rfloor$. Moreover, each of the red nodes is fully loaded. Therefore the number of red nodes is no more than $\texttt{cost}(\sigma_{in})$. There are at most d neighbours of a red node; thus the total number of brown nodes is at most $d \cdot \texttt{cost}(\sigma_{in})$. This implies that $\texttt{cost}_{\sigma_s}(R) \leq (d+1) \cdot \texttt{cost}(\sigma_{in})$. As the extent of openness of the remaining nodes is untouched, it follows that $\texttt{cost}_{\sigma_s}(P) \leq \texttt{cost}(\sigma_{in})$. ◀

3.2 Bounded Tree-width DAGs: Stable to Integrally Open Solutions

In this section, we show how to transform any stable solution σ_s for a bounded tree-width DAG into an integrally open solution σ_{io} as captured in the following Lemma.

▶ **Lemma 9.** *Let R and P respectively be the rich and poor nodes of a stable solution σ_s of a DAG, G, of constant tree-width t. Then σ_s can be converted to an integrally open solution σ_{io} such that the nodes of R remain untouched, i. e., $y_{\sigma_{io}}(u) = y_{\sigma_s}(u)$ for all $u \in R$ and $x_{\sigma_{io}}(a,u) = x_{\sigma_s}(a,u)$ for all clients $a \in A$ and nodes $u \in R$. Thus, $\texttt{cost}_{\sigma_{io}}(R) = \texttt{cost}_{\sigma_s}(R)$. Moreover, if Z_1 is the set of of fully-opened nodes of P in σ_{io}, then $\texttt{cost}_{\sigma_{io}}(Z_1) \leq (t+1) \cdot \texttt{cost}_{\sigma_s}(P)$. The remaining nodes of P are fully closed.*

Proof Sketch. Our procedure shall color poor nodes yellow and white during its course of execution; the yellow nodes will be opened and the white nodes will be closed. At the end of the procedure all the poor nodes will be colored yellow or white thereby obtaining an integrally open solution. We shall call a node *resolved* if it is either rich or colored (yellow or white) and *unresolved* otherwise. Recall that rich nodes are already open. We shall maintain two sets, `Res` and `Unres` of the *resolved* and *unresolved* nodes respectively. Every node of the graph will either be in `Res` or in `Unres`. Initially we set `Res` $= R$ and `Unres` $= P$. These sets will be modified as we color the nodes and resolve them. We shall also say that a client is *resolved* if it is only assigned to resolved nodes; we shall call it unresolved otherwise.

Consider a tree decomposition, $\langle \{V_i | i \in I\}, T \rangle$, of the DAG G having tree-width t. Fix some piece p_r containing $Rt(G)$ to be the root of the tree decomposition and assume all edges to be directed towards p_r. We say that a client a is *critical* at piece $p \in I$ if p is the highest piece along the path to p_r such that a can be assigned to some node of V_p. We call a piece *critical* if it is critical for some client.

We process the pieces of the tree decomposition in any topological order (bottom-up); let p be the piece being processed in the current iteration. We check if there is any client, a, that is critical at p. If not, we do nothing. Otherwise, consider the partitioning of the set `Unres` based on whether a node appears in p or not; let $Y = \texttt{Unres} \cap V_p$ and $\overline{Y} = \texttt{Unres} \setminus V_p$. Further, let X be the nodes of $\overline{Y}$ that appear in some piece q below p in the tree decomposition, i. e., $X = \{u \in \overline{Y} : u \in V_q$ and $\exists$ a path from q to p in the tree decomposition$\}$. We can show that (i) the extent to which the nodes in X and Y are open collectively is at least 1, i. e., $\sum_{v \in X \cup Y} y_s(v) \geq 1$; and (ii) all the clients assigned to nodes in X are attachable to some node of Y. We open up the nodes of Y and color them yellow. We account for the cost of fully opening these nodes of Y by charging them to the extent to which the nodes of $X \cup Y$ are open in the solution σ_s. We perform the pulling procedure on all the nodes of Y by pulling all the attachable clients from all nodes in $\overline{Y}$. Note that the pulling on nodes of Y will ensure that no clients remain assigned to the nodes in X any more. We therefore close all these nodes in X and color them white. We incur a factor $(t+1)$ loss in the process as the tree-width is t. Now, the nodes of Y are opened and the nodes of X are closed - we therefore move them from the set `Unres` to `Res`. Note that any poor node is charged at most once as it must be in `Unres` to be charged and it is moved to the set `Res` immediately after being charged. This completes the processing of the piece p. We now outline why (i) and (ii) must hold. For (i), since client a critical at p is unresolved, it can only be assigned to nodes in $X \cup Y$. Moreover $\sum_{v \in V} x_{\pi_{old}}(a, v) = 1$ and $x_{\pi_{old}}(a, v) \leq y_{\pi_{old}}(v)$ on any node v. Hence $\sum_{v \in X \cup Y} y_{\pi_{old}}(v) \geq 1$. For (ii), consider any client, b, serviced by some $u \in X$. Now b could not have become critical at any piece below p otherwise it would have been pulled to the node it became critical on. Thus b must be attachable to some node of Y in p.

We close any unresolved nodes left at the end as no clients can be assigned to them (every client has to be critical on some node). Hence we obtain an integrally open solution. ◀

3.3 BDBT-DAGs: Constant Factor Approximation Algorithm

In this section, we consider BDBT-DAGs and present a constant factor approximation algorithm. Combining Lemma 8 and 9, we can convert the optimal LP solution σ_{in} into an integrally open solution σ_{io}. The only issue with σ_{io} is that the request $r(a)$ of a client a may be split and assigned to multiple nodes. On the other hand, our problem definition requires that the request must be wholly assigned to a single node. We can address the issue using a cycle cancellation procedure described in prior work[1].

▶ **Lemma 10.** *Any integrally open solution σ_{io} can be converted into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{io})$.*

Using the cost analysis as stated in the lemmas, we see that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{io}) \leq 2 \cdot (\texttt{cost}_{\sigma_s}(R) + (t+1) \cdot \texttt{cost}_{\sigma_s}(P)) \leq 2 \cdot ((d+1) \cdot \texttt{cost}(\sigma_{in}) + (t+1) \cdot \texttt{cost}(\sigma_{in})) = 2 \cdot (d+t+2) \cdot \texttt{cost}(\sigma_{in})$.

We have thus established the following theorem.

▶ **Theorem 11.** *Any LP solution σ_{in} for a BDBT-DAG instance can be transformed into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot (d+t+2) \cdot \texttt{cost}(\sigma_{in})$, where d and t are respectively the degree bound and tree-width bound of the DAG.*

4 Replica Placement Problem on TBDBT-DAGs

The goal of this section is to design a constant factor approximation algorithm for the replica placement problem on TBDBT-DAGs. The algorithm builds on the procedure for handling

BDBT-DAGs presented earlier. Recall that the procedure for BDBT-DAGs works in two stages, wherein the first stage transforms an LP solution σ_{in} into a stable solution σ_s and the second stage transforms σ_s into an integrally open solution σ_{io}. Finally, the solution σ_{io} is converted into an integral solution using techniques from prior work. In the context of TBDBT-DAGs, it is difficult to obtain a stable solution, because these DAGs do not have bounded degree. Instead, our algorithm goes via a similar, but weaker notion of pseudo-stable solutions. The process of converting pseudo-stable solutions to integrally open solutions is also more involved and utilizes the concept of hierarchical solutions, which generalize integrally open solutions. It suffices to get hierarchical solutions, since prior work has shown how to transform such solutions to integral solutions. We next define pseudo-stable and hierarchical solutions, and outline our two transformations.

▶ **Definition 12** (Pseudo-stable solution)**.** An LP solution σ is said to be *pseudo-stable*, if the nodes can be partitioned into two sets R and P called *rich* and *poor* nodes respectively satisfying the following properties:

1. The rich nodes, R, are fully open.
2. For any poor node $u \in P$, the total extent to which the poor nodes service the clients attachable to u is less than W, i. e., $load_\sigma(\mathtt{Att}(u), P) < W$.
3. Every client is either serviced by
   a. only nodes in R.
   b. only nodes in P.
   c. nodes in both R and P. Any client a in this category should be attachable to exactly one node in S, where S is the set of all poor roots having a rich pivot as their out-neighbor, i. e., $S = \{u \in \mathtt{Roots(G)} \cap \mathtt{P} : \text{out-neighbor of } u \text{ is a rich pivot}\}$. The lone node in S to which a is attachable is called the *special root* of a.

   The clients in the first two categories are said to be *settled*, whereas those in the third category are said to be *unsettled*. ◀

We transform any given LP solution to a pseudo-stable solution using a procedure similar to the one used for obtaining stable solutions in the context of bounded degree DAGs; the transformation is discussed in Section 4.1. The next (and more sophisticated) stage of the algorithm converts a pseudo-stable solution into a hierarchical solution, defined below.

▶ **Definition 13** (Hierarchical solutions)**.** An LP solution σ is said to be *hierarchical* if every client is assigned to at most one partially-open node. ◀

Hierarchical solutions generalize the notion of integrally open solutions. In the context of designing constant factor approximation algorithm for the case of trees, prior work presented a procedure for transforming an LP solution into a hierarchical solution. For the case of TBDBT-DAGs, we present a procedure for transforming pseudo-stable solutions into hierarchical solutions. Given a pseudo-stable solution σ_{ps}, our procedure works by segregating σ_{ps} into two parts σ_1 and σ_2 with the following properties:

- σ_1 is a stable solution for a subset of clients. We can transform this partial solution into an integrally open solution σ_1' using the procedure in Lemma 9, since TBDBT-DAGs have bounded tree-width.
- σ_2 is a feasible solution for the remaining set of clients and has certain nice properties that allow us to transform it into a hierarchical solution σ_2'. This transformation is based on the intuition that a TBDBT-DAG consists of a skeletal tree $\mathcal{T}$, where each node is in turn a BDBT-DAG. The skeletal tree structure allows us to use ideas from the prior work[1] in obtaining the hierarchical solution σ_2'.

We finally merge σ_1' and σ_2' into a single hierarchical solution σ_h servicing all the clients. The rest of the section is devoted to describing the different transformations.

4.1 Obtaining a Pseudo-stable Solution

In this section, we show how to transform any feasible solution σ_{in} for a TBDBT-DAG into a pseudo-stable solution σ_{ps}. The following Lemma formally captures the transformation.

▶ **Lemma 14.** *Any LP solution σ_{in} can be converted into a pseudo-stable solution σ_{ps} such that $\texttt{cost}_{\sigma_{ps}}(R) \leq (d+2) \cdot \texttt{cost}(\sigma_{in})$ and $\texttt{cost}_{\sigma_{ps}}(P) \leq \texttt{cost}(\sigma_{in})$ where R and P are respectively the rich and poor nodes of the stable solution σ_{ps}.*

Proof Sketch. This transformation is a minor modification from that used in Lemma 8 for BDBT-DAGs. The transformation is divided into the *reddening* and *browning* phases.

The reddening phase is same as before. We make as many nodes fully loaded (red) as possible, and open them. We perform the browning phase with slight modifications. For every red node we color all its non-red neighbors brown except for the in-neighbors that are in $\texttt{Roots(G)}$. Note that every node has bounded degree ignoring the in-neighbors in $\texttt{Roots(G)}$; a node may have an arbitrary number of in-neighbors from $\texttt{Roots(G)}$.

We now show that the solution is pseudo-stable. We take the set of all red and brown nodes as R and the remaining nodes as P. Consider any client, a, assigned to a node v_1 in R as well as a node v_2 in P; we need to show that a is attachable to exactly one node in S. We first observe that a cannot be attachable to any brown node since then the brown node must have pulled the assignments of a from nodes in P and a would not be assigned to v_2. This also implies that v_1 is red. We now show that a is attachable to at least one node in S. Consider the path from a to v_1. The non-red node closest to v_1 on this path, say v_3, must either be brown or in $\texttt{Roots(G)} \cap \texttt{P}$. But since a is not attachable to any brown node, $v_3 \in \texttt{Roots(G)} \cap \texttt{P}$. Moreover, the out-neighbor of v_3 is a red (and hence rich) pivot and therefore $v_3 \in S$. Next we show that a cannot be attachable to two nodes in S. Suppose a is attachable to two nodes, $u_1, u_2 \in S$. Then there must be a path between them; w.l.o.g. let there be a path from u_1 to u_2. The out-neighbor of u_1 is a rich pivot, say v. Then, either v is itself brown or there exists a brown node on the path from v to u_2 (as v is red, $u_2 \in P$ and we color all non-red out-neighbors of red nodes brown). This contradicts our inference that a cannot be attachable to a brown node. Hence the solution is pseudo-stable.

The cost analysis is similar to that in Lemma 8. The change in factor arises because earlier we could color at most d neighbors brown, but now we may color brown one more neighbour – the out-neighbor of a red root (which is a pivot not in the BDBT-DAG). ◀

4.2 Obtaining a Hierarchical Solution

In this section, we show how transform a pseudo-stable solution σ_{ps} into a hierarchical solution σ_h and prove the following lemma.

▶ **Lemma 15.** *Let R and P respectively be the rich and poor nodes of a pseudo-stable solution σ_{ps} of a TBDBT-DAG, G. Let t be the maximum tree-width of any of its component BDBT-DAGs. Then σ_{ps} can be converted to a hierarchical solution σ_h such that the nodes of R remain untouched, i. e., $y_{\sigma_h}(u) = y_{\sigma_{ps}}(u)$ for all $u \in R$ and $x_{\sigma_h}(a,u) = x_{\sigma_{ps}}(a,u)$ for all clients $a \in A$ and nodes $u \in R$. Thus, $\texttt{cost}_{\sigma_h}(R) = \texttt{cost}_{\sigma_{ps}}(R)$. Moreover, $\texttt{cost}_{\sigma_h}(P) \leq (\max\{t,1\}+2) \cdot \texttt{cost}_{\sigma_s}(P)$.*

Proof. Let A_1 and A_2 be the set of settled and unsettled clients with respect to σ_{ps}. We split the original problem instance into two instances focusing on the settled and unsettled clients, respectively. This is achieved by taking two copies of the original DAG, denoted I_1 and I_2; we set $r(a) = 0$ for all unsettled clients in I_1 and $r(a) = 0$ for all settled clients in I_2. From σ_{ps}, we can get two feasible LP solutions σ_1 and σ_2 for the instances I_1 and I_2, respectively. The solutions σ_1 and σ_2 are copies of σ_{ps}, except that we set $x_{\sigma_1}(a, u) = 0$ for all unsettled clients a and $x_{\sigma_2}(a, u) = 0$ for all settled clients a, for all nodes $u \in V$.

Note that σ_1 is a stable solution for I_1. Hence, the solution σ_1 can be transformed into an integrally open solution σ'_1 for the instance I_1, using the procedure given in Lemma 9 (since a TBDBT-DAG has tree width $\max\{t, 1\}$, where t is the maximum tree-width of any of its component BDBT-DAGs). Let Z_1 be the nodes of P opened by this algorithm.

We now focus on the second instance I_2, consisting of only unsettled clients, and transform the solution σ_2 into a hierarchical solution σ'_2 using the following Lemma.

▶ **Lemma 16.** *Let R and P respectively be the rich and poor nodes of a pseudo-stable solution σ_2 of a TBDBT-DAG with constant tree-width t having only unsettled clients. Then σ_2 can be converted to a hierarchical solution σ'_2 such that the nodes of R remain untouched, i. e., $y_{\sigma'_2}(u) = y_{\sigma_2}(u)$ for all $u \in R$ and $x_{\sigma'_2}(a, u) = x_{\sigma_2}(a, u)$ for all clients a and nodes $u \in R$. Thus, $\texttt{cost}_{\sigma'_2}(R) = \texttt{cost}_{\sigma_2}(R)$. Moreover if Z_2 is the set of fully or partially open nodes of P in σ'_2, then $\texttt{cost}_{\sigma'_2}(Z_2) \leq \texttt{cost}_{\sigma_2}(P)$.*

Proof Sketch. Let U be the set of unsettled clients. We process the nodes that are special roots for some client iteratively in topological order (bottom-up) of the DAG. Let u be the node currently being processed. Let B be the set of clients for which u is a special root and let X_u be the set of nodes of P to which the clients of B are assigned (this may include u itself). We shall argue that all the clients assigned to X_u have the same special root, i. e., u. We shall perform the pulling procedure on u from $X_u \setminus \{u\}$. Note that all the clients in B will be reassigned to u and no clients will remain assigned to $X_u \setminus \{u\}$. We shall close down all the nodes of X_u and open u to the extent $\min\{\texttt{cost}_{\sigma_2}(X_u), 1\}$; thus the solution remains feasible. We shall account for the cost of opening u by charging the extent to which the nodes of X_u are open in the solution σ_2. This completes the processing of u. We now outline why all the clients assigned to X_u have u as their special root. Consider any client b assigned to a node v in X_u. By definition of X_u, there must be a client, c, assigned to v and having special root as u. It can be argued that clients attachable to the same node will have the same special root (this follows from the fact that the roots of the BDBT-DAGs are arranged in a tree-like manner in a TBDBT-DAG using a skeletal tree). Since c and b are both attachable to v, they must have the same special root; therefore the special root of b must also be u. Hence, all the clients assigned to X_u have u as the special root.

Note that the clients in B are not assigned to any poor node above u. Therefore u and B will not participate in any further processing (of other nodes that are special roots). Thus any node is charged at most once only. The special roots are taken as the set Z_2. ◀

We shall now combine the solutions σ'_1 and σ'_2 into a solution σ_h for the original input TBDBT-DAG as follows. Let Z_1 be the set of fully open nodes in σ'_1 and Z_2 be the nodes of P that are fully or partially open in σ'_2. The nodes of R remain untouched in both the solutions. We construct σ_h by opening all the nodes of R and Z_1. Then, we open all the nodes in $Z_2 \setminus Z_1$ to the extent that they were open in σ'_2. The assignments are retained from both the solutions σ'_1 and σ'_2 (the client sets are disjoint). Formally: (i) set $y_{\sigma_h}(u) = y_{\sigma'_1}(u)$ (which is 1) for all $u \in R \cup Z_1$; (ii) set $y_{\sigma_h}(u) = y_{\sigma'_2}(u)$ for all $u \in Z_2 \setminus Z_1$; (iii) set $x_{\sigma_h}(a, u) = x_{\sigma'_1}(a, u)$ for all settled clients a and nodes u; (iv) set $x_{\sigma_h}(a, u) = x_{\sigma'_2}(a, u)$ for all unsettled clients a

and nodes u. Note that no node in $Z_1 \cup Z_2$ can become fully loaded, because they belong to the set P. Moreover, the nodes of $Z_2 \setminus Z_1$ are also sufficiently open to service the clients assigned to them as they are not assigned any clients in the solution of the instance I_1 and the solution to the instance I_2 is feasible. The solution σ_h is hierarchical as the unsettled clients are assigned to at most one partially-open node since σ'_2 is hierarchical.

Using the cost analysis as stated in Lemmas 9 and 16, we see that $\texttt{cost}_{\sigma_h}(R) = \texttt{cost}_{\sigma_{ps}}(R)$ and $\texttt{cost}_{\sigma_h}(P) \leq \texttt{cost}_{\sigma'_1}(Z_1) + \texttt{cost}_{\sigma'_2}(Z_2) \leq (\max\{t,1\}+1) \cdot \texttt{cost}_{\sigma_1}(P) + \texttt{cost}_{\sigma_2}(P)$ $\leq (\max\{t,1\}+2) \cdot \texttt{cost}_{\sigma_{ps}}(P)$ This completes the proof of Lemma 15. ◀

4.3 TBDBT-DAGs: Constant Factor Approximation Algorithm

We now put together the different transformations and establish a constant factor approximation algorithm for TBDBT DAGs. Combining Lemma 14 and 15, we can convert the optimal LP solution σ_{in} into a hierarchical solution σ_h. A procedure for converting any hierarchical solution into an integral solution is implicit in prior work [1].

▶ **Lemma 17.** *Any hierarchical solution σ_h can be converted into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 136 \cdot \texttt{cost}(\sigma_h)$.*

The above procedure works by reducing the task to an issue of rounding LP solutions of a capacitated vertex cover instance, for which Saha and Khuller[11] present a 34-approximation. The proof is omitted.

Using the cost analysis as stated in the lemmas, we see that $\texttt{cost}(\sigma_{out}) \leq 136 \cdot \texttt{cost}(\sigma_h) = 136 \cdot (\texttt{cost}_{\sigma_h}(R) + \texttt{cost}_{\sigma_h}(P)) \leq 136 \cdot (\texttt{cost}_{\sigma_{ps}}(R) + (\max\{t,1\}+2) \cdot \texttt{cost}_{\sigma_{ps}}(P)) \leq 136 \cdot ((d+2) \cdot \texttt{cost}(\sigma_{in}) + (\max\{t,1\}+2) \cdot \texttt{cost}(\sigma_{in})) = 136 \cdot (d + \max\{t,1\}+4) \cdot \texttt{cost}(\sigma_{in})$

We have thus established the following theorem.

▶ **Theorem 18.** *Any LP solution σ_{in} for a TBDBT-DAG instance can be transformed into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 136 \cdot (d + \max\{t,1\}+4) \cdot \texttt{cost}(\sigma_{in})$, where d and t are respectively the maximum degree bound and the maximum tree-width bound of any component BDBT-DAG of the TBDBT-DAG.*

This yields a factor 680 approximation algorithm for the case of trees (substituting $d = 0$ and $t = 0$), improving upon the approximation ratio obtained in prior work[1].

References

1 S. Arora, V. T. Chakaravarthy, N. Gupta, K. Mukherjee, and Y. Sabharwal. Replica placement via capacitated vertex cover. *FSTTCS*, 2013.

2 A. Benoit, H. Larchevêque, and P. Renaud-Goud. Optimal algorithms and approximation algorithms for replica placement with distance constraints in tree networks. In *IPDPS*, pages 1022–1033, 2012.

3 A. Benoit, V. Rehn-Sonigo, and Y. Robert. Replica placement and access policies in tree networks. *IEEE Trans. on Parallel and Dist. Systems*, 19:1614–1627, 2008.

4 J. Chuzhoy and J. Naor. Covering problems with hard capacities. *SIAM Journal Computing*, 36(2):498–515, 2006.

5 I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. *Computer Networks*, 40:205–218, 2002.

6 U. Feige. A threshold of ln n for approximating set cover. *JACM*, 45(4):634–652, 1998.

7 R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *JCSS*, 72(1), 2006.

8 K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Trans. on Parallel and Dist. Sys.*, 12:628–637, 2001.

9 M. J. Kao and C. S. Liao. Capacitated domination problem. *Algorithmica*, pages 1–27, 2009.

10 Y. F. Lin, P. Liu, and J. J. Wu. Optimal placement of replicas in data grid environments with locality assurance. In *ICPADS*, 2006.

11 B. Saha and S. Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *ICALP*, 2012.

12 L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

13 J. J. Wu, Y. F. Lin, and P. Liu. Optimal replica placement in hierarchical data grids with locality assurance. *J. of Parallel and Dist. Computing*, 68:1517–1538, 2008.

Maintaining Approximate Maximum Matching in an Incremental Bipartite Graph in Polylogarithmic Update Time

Manoj Gupta*

Xerox Research, India
`manoj.gupta@xerox.com`, `gmanoj@cse.iitd.ac.in`

Abstract

A sparse subgraph G' of G is called a matching sparsifier if the size or weight of matching in G' is approximately equal to the size or weight of maximum matching in G. Recently, algorithms have been developed to find matching sparsifiers in a static bipartite graph. In this paper, we show that we can find matching sparsifier even in an incremental bipartite graph.

This observation leads to following results:

1. We design an algorithm that maintains a $(1+\epsilon)$ approximate matching in an incremental bipartite graph in $O(\frac{\log^2 n}{\epsilon^4})$ update time.
2. For weighted graphs, we design an algorithm that maintains $(1+\epsilon)$ approximate weighted matching in $O(\frac{\log n \log(nN)}{\epsilon^4})$ update time where N is the maximum weight of any edge in the graph.

1998 ACM Subject Classification E.1 [Data Structures]: Graphs and Networks, F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems, G.2.2 [Graph Theory]: Graph Algorithms

Keywords and phrases Graph Algorithm, Dynamic Graph

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.227

1 Introduction

A matching is a set of vertex disjoint edges in the graph. Finding a matching of maximum size in a graph is one of the most important question in combinatorial optimization. For static graph, Hopcroft and Karp [12] designed an algorithm that found maximum matching in a bipartite graph in $O(m\sqrt{n})$ time. However, extending this result to general graph turned out to be a challenging problem. Micali and Vazirani [14, 18] were the first to show that a maximum matching in a general graph can be found in $O(m\sqrt{n})$ time.

In recent years, there has been a lot of activity for maintaining approximate/exact matching in a *dynamic* graph. In a dynamic graph, at each update step an edge is added or deleted from the graph. If only insertions are allowed, then the graph is said to be an *incremental* dynamic graph. If only deletions are allowed, then the graph is said to be a *decremental* dynamic graph. If both insertions and deletions are allowed, then the graph is said to be *fully* dynamic graph.

For the analysis, we assume that an adversary executes a sequence of addition and deletion of edges in a graph with the objective of maximizing the update time of a given algorithm. An adversary is *oblivious* if he/she knows the code of the algorithm but does

* The work was done when the author was a student at IIT Delhi.

not have access to the random bits used in the algorithm. In the literature, there are some randomized algorithm that assumes an *oblivious* adversary which means that the adversary has no knowledge of the matched edges maintained by the algorithm. If the algorithm is *deterministic*, then the adversary can run this algorithm on an input sequence and find the matched edges at each update step of the algorithm. In the following literature survey, all the randomized algorithm assume *oblivious* adversary model.

Ivković and Llyod[13] were the first to investigate matching in dynamic graphs. They designed a deterministic algorithm that maintains a maximal matching with $O((n+m)^{\frac{\sqrt{2}}{2}})$ update time. Sankowski[17] designed a deterministic algorithm that maintains maximum matching in $O(n^{1.495})$ update time. Onak and Rubinfeld[16] designed a randomized algorithm that maintains a c-approximation of maximum matching in $O(\log^2 n)$ update time, where c is a large unspecified constant. Baswana, Gupta and Sen[5] showed that maximal matching can be maintained in a dynamic graph in an amortized $O(\log n)$ update time with high probability. Subsequently, Anand et al. {[2, 3]} extended this work to the weighted case, and designed a randomized algorithm that maintains a matching with a weight that is *expected* to be at least $1/4.9108 \approx 0.2036$ of the optimum. Neiman and Solomon[15] designed a deterministic algorithm that maintains a matching of size at least $2/3$ of the size of optimum matching in $O(\sqrt{m})$ time per update in general graphs. Gupta and Peng [11] generalized this result — they designed a deterministic algorithm that maintains a $(1+\epsilon)$ approximate maximum matching in $O(\sqrt{m}\epsilon^{-2})$ update time. They also extended this result to a weighted graph by designing a deterministic algorithm that maintains a $(1+\epsilon)$-approximate weighted matching in $O(m\epsilon^{-O(1/\epsilon)}\log N)$ update time where the edges have weights in the range $[1, N]$.

We investigate the problem of finding maximum matching in an incremental bipartite graph. To the best of our knowledge, there are no results for maintaining near approximate maximum matching in incremental bipartite graphs. However, the results of Gupta and Peng [11] also applies to a bipartite graph. Note that their result applies for matching in *fully* dynamic graph which seems to be a harder problem than maintaining matching in an incremental graph. However, the running time they obtain have a dependence on $(\sqrt{m})$ where m is the maximum number of edges in the graph at any point of time.

To obtain better bounds, we look for inspiration from the field of streaming algorithms. Recently, there has been much interest[9, 10, 8] in the study of graph problems in a *semi-streaming* environment. In this model, the algorithm has to work with $O(n \, poly \log n)$ space and the one of the aim of the algorithm is to minimize the total number of passes over the stream. Ahn and Guha [1] showed that there exists a semi-streaming algorithm that find a $(1+\epsilon)$-approximate bipartite matching in an unweighted/weighted graph in $O(\log n/\epsilon^3)$ passes. One of the ingredients they use is what we call as *matching sparsifier.*

► **Definition 1.** A subgraph G' of G is said to be a (ϵ, β)-sparsifier if the size or weight of matching in G' is at least $\frac{\beta}{1+\epsilon}$ whenever the size or weight of maximum matching in G is equal to β.

Ahn and Guha [1] showed that a $(O(\epsilon), \beta)$-sparsifier can be found in $O(m \log n/\epsilon^3)$ time. We show that this algorithm works even for an incremental bipartite graph. This observation helps us in showing the following results:

► **Theorem 2.** *For any $\epsilon \leq 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$ approximate maximum cardinality matching in an incremental bipartite graph in an amortized $O(\frac{\log^2 n}{\epsilon^4})$ update time.*

We extend the above result to weighted graphs:

▶ **Theorem 3.** *For any $\epsilon \leq 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$ approximate maximum weighted matching in an incremental weighted bipartite graph in an amortized $O(\frac{\log n \log(nN)}{\epsilon^4})$ update time where each edge has weight in the range $[1, N]$.*

2 Preliminaries

An undirected graph is represented by $G = (V, E)$, where V represents the set of vertices and E represents the set of edges in the graph. We will use n to denote the number of vertices $|V|$ and m to denote the number of edges $|E|$.

A *matching* in a graph is a set of independent edges in the graph. The maximum cardinality matching(MCM) in a graph is the matching of maximum size. Let $\mathcal{M}$ denote a maximum matching in the graph. Similarly, given a set of weights $w : E \rightarrow [1, N]$, we can denote the weight of a matching M as $w(M) = \sum_{e \in M} w(e)$. The maximum weight matching(MWM) in a graph is in turn the matching of maximum weight.

For measuring the quality of approximate matching, we will use the notation of α-approximation, which indicates that the objective (either cardinality or weight) given by the current solution is at least $1/\alpha$ of the optimum. Specifically, a matching M is called α-MCM if $|M| \geq \frac{1}{\alpha}$(size of MCM), and α-MWM if $w(M) \geq \frac{1}{\alpha}$(weight of MWM).

Finding or approximating MCMs and MWMs in the static setting have been intensely studied. Near linear time algorithms have been developed for finding $(1+\epsilon)$ approximations and we will make crucial use of these algorithms in our data structure. For maximum cardinality matching, such an algorithm for bipartite graph was introduced by Hopcroft and Karp[12].

▶ **Lemma 4.** [12] *For any $\epsilon < 1$, there exists an algorithm* ApproxMCM *that finds a $(1+\epsilon)$-MCM in a static unweighted bipartite graph G in $O(m\epsilon^{-1})$ time where there are m edge in G.*

For approximate MWM, there has been some recent progress. Duan et al.[6, 7] designed an algorithm that finds a $(1+\epsilon)$ approximate maximum weighted matching in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ time.

▶ **Lemma 5.** [6, 7] *For any $\epsilon < 1$, there exists an algorithm* ApproxMWM *that finds a $(1+\epsilon)$-MWM in a static weighted graph G in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ time where there are m edge in G.*

In Section 3, we reproduce the results of [1] that finds a $(O(\epsilon), \beta)$-sparsifier in a static bipartite graph. In section 4, we design an algorithm that finds a $(O(\epsilon), \beta)$-sparsifier in an incremental bipartite graph and use this algorithm to maintain a $(1+\epsilon)$-MCM. In Section 5, we reproduce the algorithm in [1] that finds a $(O(\epsilon), \beta)$-sparsifier in a static weighted bipartite graph. In Section 6, we design an algorithm that finds a $(O(\epsilon), \beta)$-sparsifier in an incremental weighted bipartite graph and use this algorithm to maintain a $(1+\epsilon)$-MWM.

3 Background

Consider the following primal-dual for bipartite matching. Here the dual(LP2) is the linear program for bipartite matching and the primal(LP1) is the linear program for vertex cover.

```
$u_i^1 = 1 \ \forall i \in V$;
for $t = 1$ to $T$ do
    Call FIND-ADMISSIBLE-SOLUTION$(t)$
    Let $M(i, \mathbf{y^t}) = \sum_{j:(i,j)\in E} y_{ij} - 1 \ \forall i$
    $\forall i$, set $u_i^{t+1} = \begin{cases} u_i^t(1+\epsilon)^{M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) \geq 0 \\ u_i^t(1-\epsilon)^{-M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) < 0 \end{cases}$
Output $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$;
```

Figure 1 MULTIPLICATIVE-WEIGHT-UPDATE(): The Multiplicative weight update method.

Primal

$$\begin{array}{llll} \min & \sum_i x_i & & \\ \text{s.t.} & x_i + x_j \geq 1 & \forall (i,j) \in E & \text{LP1} \\ & x_i \geq 0 & & \end{array}$$

Dual

$$\begin{array}{llll} \max & \sum_i y_{ij} & & \\ \text{s.t.} & \sum_{j:(i,j)\in E} y_{ij} \leq 1 & \forall i \in V & \text{LP2} \\ & y_{ij} \geq 0 & & \end{array}$$

We follow the algorithm of Ahn and Guha [1] in this section. We will use the multiplicative weight update method of Arora, Hazan and Kale [4]. Consider the multiplicative weight update method MULTIPLICATIVE-WEIGHT-UPDATE in Figure 1. The aim of MULTIPLICATIVE-WEIGHT-UPDATE is to find a feasible solution $\mathbf{y}$ such that $\sum \mathbf{y}_{ij} \approx \alpha$, i.e, the aim of our algorithm is to find a matching of a size approximately α. The algorithm runs for T iterations where we will calculate T at the end of the analysis. For each constraint associated with a vertex i in the dual LP, we associate it with a value u_i. Initially $u_i^1 = 1$. In each iteration of the algorithm, we find an *admissible* dual solution (by calling the procedure FIND-ADMISSIBLE-SOLUTION). We will define the notion of *admissibility* in Definition 6. Let $\mathbf{y^t}$ be an *admissible* solution in iteration t. Define $M(i, \mathbf{y}^t) = \sum_{j:(i,j)\in E} y_{ij}^t - 1$ for each constraint associated with vertex i. Then comes the step which justifies the name of the method, i.e., we update the weights of each u_i based on the value of $M(i, \mathbf{y^t})$. Note that ρ and ϵ are parameters in this update step which we will calculate in the analysis.

We now define the notion of *admissibility.*

▶ **Definition 6.** Define $E(\mathbf{y^t})$ to be the expected value of $M(i, \mathbf{y^t})$ if constraint i is chosen with probability $\frac{u_i^t}{\sum_j u_j^t}$, i.e, $E(\mathbf{y^t}) = \sum_i \frac{u_i^t}{\sum_j u_j^t} M(i, \mathbf{y^t})$. The dual solution $\mathbf{y^t}$ is admissible if $E(\mathbf{y^t}) \leq \delta$ and $\sum_{i,j} y_{ij} \geq \alpha$ and $M(i, \mathbf{y^t}) \in [l, \rho]$ where δ, ρ, l are parameter dependent on ϵ.

In [1], the following theorem calculates the number of iterations for which MULTIPLICATIVE-WEIGHT-UPDATE needs to run.

▶ **Theorem 7.** *Let $\epsilon \leq 1/2, \delta = 4\epsilon l$. If* FIND-ADMISSIBLE-SOLUTION *returns an* admissible *solution in all $T = \frac{2\rho \log n}{\delta\epsilon}$ iterations , then for all constraints $i, M(i, \mathbf{y}) \leq 1$*

Theorem 7 implies that if FIND-ADMISSIBLE-SOLUTION returns an *admissible* solution for T iteration, then we can find a feasible solution $\mathbf{y}$. Since $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$ and $\sum_{ij} y_{ij}^t \geq \alpha$, $\sum_{ij} y_{ij} \geq \left(\frac{1}{1+4\delta}\right)\alpha$. So we obtain a feasible *fractional* solution of a value approximately equal to α.

```
∀i, let $x_i^t = \frac{\alpha u_i^t}{\sum_j u_j^t}$ ;
Let $E_{violated}^t = \{(i,j) : x_i^t + x_j^t < 1\}$
Find a maximal matching $S_t$ in $E_{violated}^t$.
if $|S_t| < \delta\alpha$ then
    For each $(i,j) \in S_t$, increase $x_i^t$ and $x_j^t$ by 1
    return failure
else
    Return $y_{ij}^t = \frac{\alpha}{|S_t|}$ for $(i,j) \in S_t$ and 0 otherwise
```

Figure 2 FIND-ADMISSIBLE-SOLUTION(t): The procedure that finds an *admissible* solution.

3.1 MCM

The multiplicative weight update method mandates that an *admissible* solution is found at each iteration. We now reproduce FIND-ADMISSIBLE-SOLUTION designed by Ahn and Guha [1] which finds an *admissible* solution.

FIND-ADMISSIBLE-SOLUTION starts by setting x_i values for all vertices in the graph. The value of x_i^t is α times the probability of choosing a vertex i under the probability distribution $\mathbf{u^t}$. We look at the edges which violated the dual LP constraint using the assignment $\mathbf{x^t}$, i.e., all the edges $e = (i,j)$ such that $x_i^t + x_j^t < 1$. Let the set of all violated edge be denoted by $E_{violated}$. We find a maximal matching S_t in the set $E_{violated}$. If $|S_t| < \delta\alpha$, we return a failure, else we return $\mathbf{y^t}$ by setting $y_{ij}^t = \alpha/|S_t|$ for all the edges in the matching.

We reproduce the following lemma from [1]

▶ **Lemma 8.** [1] *If $|S_t| < \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *returns a feasible solution for LP 2 with a value at most $(1+2\delta)\alpha$.*

Proof. For each edge (i,j) such that $x_i^t + x_j^t < 1$, at least one of the endpoint is in the maximal matching (since S_t is a maximal matching). We increase x_i^t and x_j^t by 1 to satisfy this constraint. So all the violated constraints are satisfied. Initially, $\sum_i x_i^t = \alpha$ and since we have increased the value of all the vertices in the maximal matching, the total increase is $< 2\delta\alpha$. So the total value of the solution of LP1 is at most $(1+2\delta)\alpha$. ◀

▶ **Lemma 9.** [1] *If $|S_t| \geq \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *returns an* admissible *solution with $l = 1$ and $\rho = 1/\delta$ and $E(\mathbf{y^t}) \leq \delta$.*

Proof. Since $y_{i,j}^t = \alpha/|S_t|$, for all $(i,j) \in S_t$, so

$$\begin{array}{lcll}
\sum_{(i,j)\in S_t} y_{i,j}^t & = & \alpha & \\
\sum_{(i,j)\in S_t} y_{i,j}^t(x_i^t + x_j^t) & < & \alpha & \{ \text{ For each } (i,j) \in S_t,\ x_i^t + x_j^t < 1 \ \} \\
\sum_{(i,j)\in E} y_{i,j}^t(x_i^t + x_j^t) & = & \alpha & \{ \text{ Since } y_{i,j}^t = 0 \text{ for all other edges } \} \\
\sum_i x_i^t \sum_{j:(i,j)\in E} y_{i,j}^t & = & \sum_i x_i^t & \\
\sum_i x_i^t \big(\sum_{j:(i,j)\in E} y_{i,j}^t - 1 \big) & = & 0 & \\
\sum_i x_i^t M(i, y^t) & = & 0 & \\
\sum_i \frac{x_i^t}{\sum_j x_j^t} M(i, y^t) & = & 0 &
\end{array}$$

This implies $E(\mathbf{y^t}) \leq 0 \leq \delta$. Also, if $(i,j) \in S_t$, $y_{ij}^t = 1/\delta$, this implies $M(i, \mathbf{y^t}) = 1/\delta$, so $\rho = 1/\delta$. If $(i,j) \notin S_t$, $M(i, \mathbf{y^t}) = -1$, so $l = 1$. ◀

Now comes the crucial step of the algorithm. We never want FIND-ADMISSIBLE-SOLUTION to fail. This implies that the size of the maximal matching found by our algorithm should always be greater than or equal to $\delta\alpha$. To achieve this, we set a suitable value of α. Let M be *any* matching in the graph such that $|M| = \beta$. Suppose that we set $\alpha_i = (1+\epsilon)^i$ for $i \geq 0$. Let α_j be the smallest value above β. So $\alpha_j \geq \beta \geq \alpha_{j-1}$. We now prove the following lemma:

▶ **Lemma 10.** *If $\beta \geq \alpha_{j-1}$ and if we set $\alpha = \alpha_j/(1+\epsilon)^9$ in* MULTIPLICATIVE-WEIGHT-UPDATE *procedure, then* FIND-ADMISSIBLE-SOLUTION *never fails.*

Proof. Suppose the algorithm fails. By Lemma 8, this implies that there exists a feasible solution of LP1 with value $(1+2\delta)\alpha$. Since $\epsilon = \delta/4$ (since $\delta = 4\epsilon l$ and $l = 1$), this value is $\leq (1+8\epsilon)\alpha = (1+8\epsilon)\alpha_j/(1+\epsilon)^9 \leq (1+8\epsilon)\beta/(1+\epsilon)^8 < \beta$. This leads to a contradiction since the minimum value of LP1 is $\geq \beta$ (as the size of maximum matching is $\geq \beta$). This implies that FIND-ADMISSIBLE-SOLUTION never fails. ◀

Using Theorem 7, if we set $\alpha = \alpha_j/(1+\epsilon)^9$, then after T iterations, we will find a feasible *fractional* solution $\mathbf{y}$. In MULTIPLICATIVE-WEIGHT-UPDATE procedure, we set $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$. Since FIND-ADMISSIBLE-SOLUTION returns an *admissible* solution of value $= \alpha$ in each iteration, we have $\sum y_{ij}^t = \left(\frac{1}{1+4\delta}\right)\alpha$. Since $\delta = 4\epsilon l$ and $l = 1$, $\delta = 4\epsilon$. So,

$$\begin{aligned}
\sum \mathbf{y}_{ij} &= \left(\tfrac{1}{1+16\epsilon}\right)\alpha \\
&= \left(\tfrac{1}{1+16\epsilon}\right)\alpha_j/(1+\epsilon)^9 \\
&\geq \tfrac{1}{(1+16\epsilon)(1+\epsilon)^9}\alpha_j \\
&\geq \tfrac{1}{(1+16\epsilon)(1+75\epsilon)}\alpha_j && \left(\text{if } \epsilon \leq 1/2, \text{ then } \tfrac{1}{(1+\epsilon)^9} \geq \tfrac{1}{1+75\epsilon}\right) \\
&\geq \tfrac{1}{(1+1200\epsilon)}\alpha_j \\
&\geq \tfrac{1}{(1+1200\epsilon)}\beta && \left(\text{since } \beta \leq \alpha_j\right)
\end{aligned}$$

Thus, we have proved the following lemma:

▶ **Lemma 11.** *If* FIND-ADMISSIBLE-SOLUTION *returns an* admissible *solution for T iterations, then the size of the fractional matching returned by* MULTIPLICATIVE-WEIGHT-UPDATE *is* $\geq \frac{1}{1+1200\epsilon}\alpha_j \geq \frac{1}{(1+1200\epsilon)}\beta$

So if we set $\epsilon' = 1/1200\epsilon$, we get a feasible solution $\mathbf{y}$ such that the size of this fractional solution $\mathbf{y}$ is $\geq (1-\epsilon')\beta$. Also note that at each iteration t, FIND-ADMISSIBLE-SOLUTION selects at most n edges and sets $y_{ij}^t > 0$ for these n edges. Since there are at most $T = \frac{2\rho \log n}{\delta\epsilon}$ iterations, the total number of edges selected by FIND-ADMISSIBLE-SOLUTION with $\mathbf{y}_{ij} > 0$ is $\leq \frac{2\rho \log n}{\delta\epsilon}n$. We have the following lemma:

▶ **Theorem 12.** *There exists an algorithm which finds a $(O(\epsilon), \beta)$-sparsifier G' of G of size $O(n \log n/\epsilon^3)$. Moreover, the time taken to find such a graph is $O(m \log n/\epsilon^3)$,*

Proof. From the above discussion, we claim the total number of edges selected by FIND-ADMISSIBLE-SOLUTION is $O(\frac{2\rho \log n}{\delta\epsilon}n) = O(\frac{n \log n}{\epsilon^3})$ (since $\delta = 4\epsilon$ and $\rho = 1/\delta$). By Lemma 11, the size of fractional matching in G', i.e., $\sum \mathbf{y}_{ij}$ is $\geq \frac{1}{(1+1200\epsilon)}\beta$. Using the integrality of bipartite matching polytope, we claim that the size of maximum matching in G' is $\geq \frac{1}{(1+1200\epsilon)}\beta$. Regarding the running time, note that each iteration is dominated by the running time of FIND-ADMISSIBLE-SOLUTION — which is $O(m)$. Since $\rho = 1/\delta$ and $\delta = 4\epsilon$, there are at most $T = \frac{2\rho \log n}{\delta\epsilon} = O(\frac{\log n}{\epsilon^3})$ iterations and the total running time is $O(m \log n/\epsilon^3)$. ◀

4 Incremental MCM

Overview

In this section, we show that we can find a $(O(\epsilon), \beta)$-sparsifier in an incremental bipartite graph. This observation is then used to maintain a $(1+\epsilon)$-MCM in the following way: We run many versions of the algorithm in Theorem 12 in parallel such that in the kth run, we set $\alpha = \alpha_k/(1+\epsilon)^9$ in the MULTIPLICATIVE-WEIGHT-UPDATE procedure where $\alpha_k = (1+\epsilon)^k$ and $k \geq 9$. Since the size of maximum matching is $\leq n$, $k \leq \frac{\log n}{\log(1+\epsilon)} = O(\frac{\log n}{\epsilon})$. In the kth run, we want to find a $(O(\epsilon), \alpha_k)$-sparsifier. Note that initially a $(O(\epsilon), \alpha_k)$-sparsifier may not exist as the size of maximum matching itself may be $< \alpha_k$. So our algorithm returns failure till the size of maximum matching is approximately equal to α_k. At any given update step, say l, let i be the highest numbered version for which MULTIPLICATIVE-WEIGHT-UPDATE has not failed. We find a $(1+\epsilon)$-MCM M_i in the $(O(\epsilon), \alpha^i)$-sparsifier found in the ith run. We will show that M_i is a $(1+O(\epsilon))$-MCM in G_l, i.e., the ratio between the size of M_i and the maximum matching at the lth update step is $1+O(\epsilon)$.

Consider an incremental graph where at the update step l, an edge e_l is added to the graph, i.e., the graph at lth update step $G_l = G_{l-1} \cup e_l$. We use the incremental version of MULTIPLICATIVE-WEIGHT-UPDATE and FIND-ADMISSIBLE-SOLUTION (see Figure 3 and 4).

Fix a value of k. We describe our adaptation of the algorithm in the previous section for kth run of the algorithm when $\alpha = \alpha_k/(1+\epsilon)^9$. Consider the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION. Before calling this procedure, we set $u_i^1 = 1$ for all the constraints of LP1. Since the graph is empty initially, the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION fails in the first iteration. At update step l, e_l is added to the graph G_{l-1}, and the procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE is run. The procedure finds the iteration (say t), where INCREMENTAL-FIND-ADMISSIBLE-SOLUTION has failed. Since a new edge is added to the graph, the procedure calls INCREMENTAL-FIND-ADMISSIBLE-SOLUTION hoping that there exists an *admissible* solution after the addition of this new edge. If INCREMENTAL-FIND-ADMISSIBLE-SOLUTION returns failure, then there is still no *admissible* solution found at iteration t. Else, INCREMENTAL-FIND-ADMISSIBLE-SOLUTION successfully finds an *admissible* solution. We increment t and try to find an *admissible* solution in iteration $t+1$. If $t = T+1$, then using Lemma 11, we claim that the size of maximum matching in our sparsifier is at least $\frac{1}{(1+1200\epsilon)}\alpha_k$. This implies that we have found a $(O(\epsilon), \alpha^k)$-sparsifier at this update step. We then run APPROXMCM on this $(O(\epsilon), \alpha^k)$-sparsifier. We then *stop* the kth run of our algorithm.

We now describe the incremental version of the algorithm FIND-ADMISSIBLE-SOLUTION (see Figure 4). If procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE calls procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION for the first time in the iteration t, then we initialize all x_u^t's and find a maximal matching S_t as in procedure FIND-ADMISSIBLE-SOLUTION. Else we need to update S_t with respect to this newly added edge $e_l = (u, v)$. If $x_u^t + x_v^t$ is less than 1 and u and v are not adjacent to any edge in S_t, then the edge e_l is added to S_t. If $S_t < \delta\alpha$, then FIND-ADMISSIBLE-SOLUTION was unable to find an *admissible* solution and returns failure. Else we return an *admissible* solution $\mathbf{y^t}$(this part is same as in the FIND-ADMISSIBLE-SOLUTION). The important thing to note is that we return an *admissible* solution as soon as $|S_t|$ is equal to $\delta\alpha$.

We now prove the following lemma:

▶ **Lemma 13.** *If the size of maximum matching crosses α_{k-1} at update step l, then the kth run of* INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE stops *before or at update step l.*

```
Let the current update step be $l$ with $e_l$ being added to graph $G_{l-1}$.
Let $t$ be the iteration in which INCREMENTAL-FIND-ADMISSIBLE-SOLUTION has
previously failed.
while INCREMENTAL-FIND-ADMISSIBLE-SOLUTION(t) does not return failure do
    $t \leftarrow t+1$
    if $t$ = T+1 then
        Run ApproxMCM on the sparsifier found by the $k$th run
        Stop the $k$th run of the algorithm
    else
        Let $M(i,\mathbf{y^t}) = \sum_{j:(i,j)\in E} y_{ij} - 1 \ \forall i$
        $\forall i$, set $u_i^{t+1} = \begin{cases} u_i^t(1+\epsilon)^{M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) \geq 0 \\ u_i^t(1-\epsilon)^{-M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) < 0 \end{cases}$
```

Figure 3 INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE(): The incremental version of Multiplicative Weight Update Method.

```
if this is the first call to INCREMENTAL-FIND-ADMISSIBLE-SOLUTION in iteration $t$ then
    $\forall w$, let $x_w^t = \frac{\alpha u_w^t}{\sum_j u_j^t}$;
    Let $E^t_{violated} = \{(u,v) : x_u^t + x_v^t < 1\}$
    Find a maximal matching $S_t$ in $E^t_{violated}$.
else
    if $x_u^t + x_v^t < 1$ and $u$ and $v$ are free with respect to $S_t$ then
        $S_t \leftarrow S_t \cup e_i$
if $|S_t| < \delta\alpha$ then
    return failure
else
    Return $y_{ij}^t = \frac{\alpha}{|S|}$ for $(i,j) \in S_t$ and 0 otherwise
```

Figure 4 INCREMENTAL-FIND-ADMISSIBLE-SOLUTION(t): The incremental version FIND-ADMISSIBLE-SOLUTION that finds an *admissible* solution.

Proof. Suppose that the procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE does not stop at or before the update step l. This implies that there exists an iteration t at which INCREMENTAL-FIND-ADMISSIBLE-SOLUTION is unable to find an *admissible* solution. Note that INCREMENTAL-FIND-ADMISSIBLE-SOLUTION incrementally maintains a maximal matching S_t. This implies that $|S_t| < \delta\alpha$. At the lth update step, the size of maximum matching is $\geq \alpha_{k-1}$, so Lemma 10 mandates that FIND-ADMISSIBLE-SOLUTION (and therefore INCREMENTAL-FIND-ADMISSIBLE-SOLUTION) never fails. This lead to a contradiction thus proving the lemma. ◀

At an update step l, let i be the highest numbered version for which the procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE has stopped. This implies that the size of maximum matching at this update step is less than α_i — if not then by Lemma 13, even $(i+1)$th run of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE should have stopped. After the ith run stops, we use ApproxMCM to find a $(1+\epsilon)$-MCM in the sparsifier found at the ith run. Using Lemma 11, we claim that the size of matching in the sparsifier is $\frac{1}{(1+1200\epsilon)}\alpha^i$.

```
$\forall i$, let $x_i^t = \frac{\alpha u_i^t}{\sum_j u_j^t}$;
Let $E^t_{violated,k} = \{(i,j) : x_i^t + x_j^t < w_{ij}, \alpha/2^k \le w_{ij} \le \alpha/2^{k-1}\}$
Find a maximal matching $S_t^k$ in $E^t_{violated,k}$ for each $k = 1, 2, \ldots, \lceil \log \frac{n}{\delta} \rceil = O(\log n)$.
Let $S_t = \cup_k S_k$, $\Delta = w(S_t)$
if  $\Delta < \delta\alpha$ then
    For each $(i,j) \in S_t$, increase $x_i$ and $x_j$ by $2w_{ij}$.
    Further increase every $x_i$ by $\delta\alpha$
    Return $\mathbf{x}$ and report failure.
else
    $S' \leftarrow \emptyset$
    while  $S_t \neq \emptyset$ do
        Pick the heaviest edge $(i,j)$ from $S_t$ and add it to $S'$
        Remove all the edges adjacent to $i$ and $j$ from $S_t$
    Return $y_{ij}^t = \frac{\alpha}{w(S')}$ for $(i,j) \in S'$ and 0 otherwise
```

Figure 5 FIND-ADMISSIBLE-SOLUTION(t): The oracle which finds an *admissible* solution.

So, APPROXMCM finds a matching of size at least $\frac{\alpha^i}{(1+1200\epsilon)(1+\epsilon)} \ge \frac{1}{(1+1202\epsilon)}\alpha^i$. If we use the matching obtained at the ith run as our current matching, the approximation ratio of our matching with respect to the maximum matching is $\le \frac{\alpha_i}{\alpha_i/(1+1202\epsilon)} = 1 + 1202\epsilon$.

Now we analyze the running time of the kth run of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE. The running time of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE is dominated by the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION. We claim that the running time of this procedure at the tth iteration is at most $O(m)$ where m is the number of edges at the end of all updates. This is true because the initialization step takes at most $O(m)$ time and processing each update e_l takes $O(1)$ time. Since there are $T = O(\log n/\epsilon^3)$ iterations, the total running time for the kth run of our algorithm is $O(m \log n/\epsilon^3)$.

Since we run $O(\log n/\epsilon)$ versions of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE in parallel, the total time taken is $O(m \log^2 n/\epsilon^4)$. This implies an amortized update time of $O(\frac{\log^2 n}{\epsilon^4})$. Thus, we have proved the following theorem:

▶ **Theorem 14.** *For any $\epsilon \le 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$-MCM in an incremental bipartite graph in amortized $O(\frac{\log^2 n}{\epsilon^4})$ update time.*

5 MWM

In this section, we follow the algorithm in [1] that finds a $(1+\epsilon)$-MWM in a static weighted graph. Consider the maximum weighted matching problem where each edge (u,v) has weight w_{uv} such that the weight of every edge in the graph is $\le N$. As before we use the MULTIPLICATIVE-WEIGHT-UPDATE procedure in Figure 1. If the procedure FIND-ADMISSIBLE-SOLUTION returns an admissible solution in each of the T iterations, then the procedure MULTIPLICATIVE-WEIGHT-UPDATE finds a feasible solution. We reproduce the procedure FIND-ADMISSIBLE-SOLUTION for weighted graphs from [1].

The procedure starts by partitioning the edges across $\log n$ levels. If an edge (i,j) has weight between $\alpha/2^{k-1}$ and $\alpha/2^k$, then it is at level k where $k \le \lceil \log \frac{n}{\delta} \rceil$. We ignore the edges with weight $\le \alpha/2^{\log \frac{n}{\delta}}$. We find a maximal matching at all the k levels and let S_t be the union of these matchings. Let Δ be the sum of weight of all the edges in S_t. If $\Delta < \delta\alpha$, then we report failure, else we find a matching S' from S_t. This procedure adds the heaviest edge (i, j) in S_t in S' and remove all the edges adjacent to i and j from S_t. The procedure then returns an *admissible* solution $\mathbf{y_t}$.

Ahn and Guha [1] proved the following:

▶ **Lemma 15.** [1] *If $\Delta > \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *returns an admissible solution with $\rho = \frac{5}{\delta}$ and $l = 1$.*

▶ **Lemma 16.** [1] *If $\Delta \leq \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *finds a feasible solution of vertex cover with value $(1 + 5\delta)\alpha$ and returns failure.*

Now comes the crucial step of the algorithm. We never want FIND-ADMISSIBLE-SOLUTION to fail. This implies that the weight of S_t found by our algorithm should always be greater than equal to $\delta\alpha$. Now we set a suitable value of α to achieve this. Let M be *any* matching in the graph such that the weight of M, $w(M) = \beta$. Suppose that we set $\alpha_i = (1 + \epsilon)^i$ for $i \geq 0$. Let α_j be the smallest value above β. So $\alpha_j \geq \beta \geq \alpha_{j-1}$. We now prove the following lemma:

▶ **Lemma 17.** *If $\beta \geq \alpha_{j-1}$ and if we set $\alpha = \alpha_j/(1+\epsilon)^{21}$, then* FIND-ADMISSIBLE-SOLUTION *never fails*

Proof. Suppose that FIND-ADMISSIBLE-SOLUTION fails. By Lemma 16, this implies that there exists a feasible solution of vertex cover with value $(1 + 5\delta)\alpha$. Since $\delta = 4\epsilon l$ and $l = 1$, this value is $\leq (1 + 20\epsilon)\alpha = (1 + 20\epsilon)\alpha_j/(1 + \epsilon)^{21} = (1 + 20\epsilon)\beta/(1 + \epsilon)^{20} < \beta$. This leads to a contradiction since the minimum weight of the vertex cover is $\geq \beta$ (as the weight of maximum matching is $\geq \beta$). This implies that the oracle FIND-ADMISSIBLE-SOLUTION never fails. ◀

Using Lemma 7, if we set $\alpha = \alpha_j/(1 + \epsilon)^{21}$, then after T iterations, we will find a feasible solution $\mathbf{y}$. In the MULTIPLICATIVE-WEIGHT-UPDATE procedure, we set $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$. Since FIND-ADMISSIBLE-SOLUTION returns an *admissible* solution of value $= \alpha$ in each iteration, we have $\sum y^t_{ij} = \left(\frac{1}{1+4\delta}\right)\alpha$. Since $\delta = 4\epsilon l$ and $l = 1$, $\delta = 4\epsilon$. So,

$$\begin{aligned}
\sum \mathbf{y}_{ij} &= \left(\tfrac{1}{1+16\epsilon}\right)\alpha \\
&= \left(\tfrac{1}{1+16\epsilon}\right)\tfrac{\alpha_j}{(1+\epsilon)^{21}} \\
&\geq \tfrac{1}{(1+16\epsilon)(1+10000\epsilon)}\alpha_j \quad \left(\text{ if } \epsilon \leq 1/2, \text{ then } \tfrac{1}{(1+\epsilon)^{21}} \geq \tfrac{1}{1+10000\epsilon}\ \right) \\
&\geq \tfrac{1}{(1+160000\epsilon)}\alpha_j \qquad (2) \\
&\geq \tfrac{1}{(1+160000\epsilon)}\beta \quad \left(\text{ since } \beta \leq \alpha_j\right)
\end{aligned}$$

We can state the following lemma:

▶ **Lemma 18.** *If* FIND-ADMISSIBLE-SOLUTION *returns an admissible solution for T iterations then the size of fractional matching return by our algorithm is $\geq \frac{1}{1+160000\epsilon}\alpha_j \geq \frac{1}{(1+160000\epsilon)}\beta$*

If we set $\epsilon' = 1/160000\epsilon$, we get a feasible solution $\mathbf{y}$ such that the size of this fractional solution $\mathbf{y}$ is $\geq (1 - \epsilon')\beta$. Also note that at each iteration t, FIND-ADMISSIBLE-SOLUTION selects at most n edges and sets $y^t_{ij} > 0$ for these n edges. Since there are at most $T = \frac{2\rho \log n}{\delta\epsilon}$ iterations, the total number of edges selected by FIND-ADMISSIBLE-SOLUTION with $\mathbf{y}_{ij} > 0$ is $\leq \frac{2\rho \log n}{\delta\epsilon}n$. Since $\rho = 5/\delta$ and $\delta = 4\epsilon l$, we have the following lemma:

▶ **Theorem 19.** *There exists an algorithm which finds a $(O(\epsilon), \beta)$-sparsifier G' of G of size $O(n \log n/\epsilon^3)$. Moreover, the time taken to find such a graph is $O(m \log n/\epsilon^3)$.*

Proof. The proof is identical to the proof of Theorem 12. ◀

6 Incremental MWM

In this section, we show that we can find a $(O(\epsilon), \beta)$-sparsifier in an incremental weighted bipartite graph where the weight of any edge is $\leq N$. This observation is then used to maintain a $(1+\epsilon)$-MWM in the following way: We run many versions of algorithm in Theorem 19 in parallel such that in the kth run, we set $\alpha = \alpha_k/(1+\epsilon)^{21}$ in the MULTIPLICATIVE-WEIGHT-UPDATE where $\alpha_k = (1+\epsilon)^k$ and $k \geq 21$. Since the size of maximum matching is $\leq nN$, $k \leq \frac{\log(nN)}{\log(1+\epsilon)} = O(\frac{\log(nN)}{\epsilon})$. In the kth run, we want to find a $(O(\epsilon), \alpha_k)$-sparsifier. Note that initially a $(O(\epsilon), \alpha_k)$-sparsifier may not exist as the size of maximum weighted matching itself may be $< \alpha_k$. So our algorithm returns failure till the size of maximum matching is approximately equal to α_k. At any given update step, say l, let i be the highest numbered version for which MULTIPLICATIVE-WEIGHT-UPDATE has not failed. We find a $(1+\epsilon)$-MWM M_i in the $(O(\epsilon), \alpha^i)$-sparsifier found in the ith run. We will show that the ratio between the weight of M_i and the weight of maximum matching at the lth update step is $1 + O(\epsilon)$.

Consider an incremental weighted graph where at the update step l, an edge $e_l = (i, j)$ with weight w_{ij} is added to the graph, i.e., the graph at lth update step $G_l = G_{l-1} \cup e_l$. We use the incremental version of MULTIPLICATIVE-WEIGHT-UPDATE in Figure 3. Now, we design an incremental version of FIND-ADMISSIBLE-SOLUTION (see Figure 6).

Fix a value of k. We describe our adaptation of the algorithm in the previous section for kth run of the algorithm when $\alpha = \alpha_k/(1+\epsilon)^{21}$. If procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE calls procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION for the first time in the iteration t, we initialize all x_u^t's and find a maximal matching S_t as in procedure FIND-ADMISSIBLE-SOLUTION. Else we need to update S_t with respect to this newly added edge $e_l = (u, v)$. If $(u, v) \in E_{violated,k}$ and u and v are free with respect to maximal matching in $E_{violated,k}$, edge e_l is added to S_t. If $S_t < \delta\alpha$, then the oracle was unable to find an *admissible* solution and returns failure. Else we return an *admissible* solution $\mathbf{y^t}$ (this part is same as in FIND-ADMISSIBLE-SOLUTION). Again note that we return an *admissible* solution as soon as $|S_t|$ is equal to $\delta\alpha$.

We now prove the following lemma:

▶ **Lemma 20.** *If the size of maximum matching crosses α_{k-1} at update step l, then the kth run of* INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE *stops before or at update step l.*

Proof. Similar to the proof of Lemma 13. ◀

Similar to our analysis in Section 4, we claim that our algorithm maintains a $(1+\epsilon)$-MWM at every update step.

Now we analyze the running time of our algorithm. Consider the ith run of the algorithm. The running time of the algorithm is dominated by the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION. We claim that the running time of this procedure at the tth iteration is at most $O(m)$ where m is the number of edges at the end of all updates. This is true because the initialization step takes at most $O(m)$ time and processing each update e_l takes $O(1)$ time. Since there are $T = O(\frac{\log n}{\epsilon^3})$ iterations, the total running time for the ith run of our algorithm is $O(m \log n/\epsilon^3)$. Since there are $O(\log(nN)/\epsilon)$ version of our algorithm, the total time taken by the algorithm is $O(m \log n \log(nN)/\epsilon^4)$. This implies an amortized update time of $O(\log n \log(nN)/\epsilon^4)$.

We claim the following theorem:

▶ **Theorem 21.** *For any $\epsilon \leq 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$-MWM in an incremental weighted bipartite graph in amortized $O(\frac{\log n \log(nN)}{\epsilon^4})$ update time where each edge has weight in the range $[1, N]$.*

```
if this is the first call to INCREMENTAL-FIND-ADMISSIBLE-SOLUTION in iteration t then
    ∀i, let $x_i^t = \frac{\alpha u_i^t}{\sum_j u_j^t}$;
    Let $E_{violated,k}^t = \{(i,j) : x_i^t + x_j^t < w_{ij}, \alpha/2^k \le w_{ij} \le \alpha/2^{k-1}\}$
    Find a maximal matching $S_t^k$ in $E_{violated,k}^t$ for each $k = 1, 2, \dots, \lceil \log \frac{n}{\delta} \rceil = O(\log n)$.
    Let $S_t = \cup_k S_k$, $\Delta = w(S_t)$
else
    if $(u, v) \in E_{violated,k}^t$ and u and v are free with respect to the maximal matching in $E_{violated,k}^t$ then
        $S_t \leftarrow S_t \cup e_l$
if $|S_t| < \delta\alpha$ then
    return failure
else
    $S' \leftarrow \emptyset$
    while $S_t \neq \emptyset$ do
        Pick the heaviest edge $(i, j)$ from S and add it to $S'$
        Remove all the edges adjacent to $i$ and $j$ from $S_t$
    Return $y_{ij}^t = \frac{\alpha}{w(S')}$ for $(i, j) \in S'$ and 0 otherwise
```

Figure 6 INCREMENTAL-FIND-ADMISSIBLE-SOLUTION(e_l, t): The incremental version of the oracle that finds an *admissible* solution.

References

1 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013.

2 Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. Maintaining Approximate Maximum Weighted Matching in Fully Dynamic Graphs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 257–266, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

3 Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. Maintaining approximate maximum weighted matching in fully dynamic graphs. *CoRR*, abs/1207.3976, 2012.

4 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

5 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 383–392. IEEE, 2011.

6 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, January 2014.

7 Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for approximate and exact maximum weight matching. *CoRR*, abs/1112.0790, 2011.

8 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1-2):490–508, June 2012.

9 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005.

10 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, December 2008.

11 Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science*, 2013.

12 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

13 Zoran Ivkovic and Errol L. Lloyd. Fully dynamic maintenance of vertex cover. In *WG '93: Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 99–111, London, UK, 1994. Springer-Verlag.

14 S. Micali and V.V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 17–27. IEEE, 1980.

15 Ofer Neiman and Shay Solomon. Deterministic algorithms for fully dynamic maximal matching. *CoRR*, abs/1207.1277, 2012.

16 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, pages 457–464, 2010.

17 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126, 2007.

18 Vijay V. Vazirani. An improved definition of blossoms and a simpler proof of the MV matching algorithm. *CoRR*, abs/1210.4594, 2012.

The Complexity of Counting Models of Linear-time Temporal Logic*

Hazem Torfah and Martin Zimmermann

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
{torfah, zimmermann}@react.uni-saarland.de

Abstract

We determine the complexity of counting models of bounded size of specifications expressed in Linear-time Temporal Logic. Counting word-models is #P-complete, if the bound is given in unary, and as hard as counting accepting runs of nondeterministic polynomial space Turing machines, if the bound is given in binary. Counting tree-models is as hard as counting accepting runs of nondeterministic exponential time Turing machines, if the bound is given in unary. For a binary encoding of the bound, the problem is at least as hard as counting accepting runs of nondeterministic exponential space Turing machines. On the other hand, it is not harder than counting accepting runs of nondeterministic doubly-exponential time Turing machines.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Model Counting, Temporal Logic, Model Checking, Counting Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.241

1 Introduction

Model counting, the problem of computing the *number of models* of a logical formula, generalizes the satisfiability problem and has diverse applications: many probabilistic inference problems, such as Bayesian net reasoning [13], and planning problems, such as computing the robustness of plans in incomplete domains [15], can be formulated as model counting problems of propositional logic. Model counting for Linear-time Temporal Logic (LTL) has been recently introduced in [8]. LTL is the most commonly used specification logic for reactive systems [16] and the standard input language for model checking [2, 5] and synthesis tools [3, 4, 6]. LTL model counting asks for computing the number of transition systems that satisfy a given LTL formula. As such a formula has either zero or infinitely many models one considers models of *bounded* size: for a formula φ and a bound k, the problem is to count the number of models of φ of size k. This is motivated by applications like bounded model checking [2] and bounded synthesis [7], where one looks for short error paths and small implementations, respectively, by iteratively increasing a bound on the size of the model. Just like propositional model counting generalizes satisfiability, by considering two types of bounded models, namely, *word-models* (of length k) and *tree-models* (of height k), the authors of [8] introduced quantitative extensions of model checking and synthesis.

Word-models are ultimately periodic words of the form $u.v^\omega$ of bounded length $|u.v|$, which are used to model computations of a system. Counting word-models can be used in *model checking* to determine not only the existence of computations that violate the specification, but also the *number* of such *violations*. To this end, one turns the model

* This work was partially supported by the German Research Foundation (DFG) as part of SFB/TR 14 AVACS and by the Deutsche Telekom Foundation.

checking problem into an LTL satisfiability problem by encoding the transition system and the negation of the specification into a single LTL formula. Its models represent erroneous computations of the system, i.e., counting them gives a quantitative notion of satisfaction.

Tree-models are finite trees (of fixed out-degree) of bounded height with back-edges at the leaves, i.e., tree-models can be exponentially-sized in the bound. They are used to describe implementations of the input-output behavior of reactive systems (see, e.g., [7]), namely the edges of a tree-model represent the input behavior of the environment and the nodes represent the corresponding output behavior of the system. In *synthesis*, counting tree-models can be used to determine not only the existence of an implementation that satisfies the specification, but also the *number* of such *implementations*. This number is a helpful metric to understand how much room for implementation choices is left by a given specification, and to estimate the impact of new requirements on the remaining design space.

For *safety* LTL specifications [17], algorithms solving the word- and the tree-model counting problem were presented in [8]. The running time of the algorithms is linear in the bound and doubly-exponential respectively triply-exponential in the length of the formula. The high complexity in the formula is, however, not a major concern in practice, since specifications are typically small while models are large (cf. the state-space explosion problem).

Here, we complement these algorithms by analyzing the computational complexity of the model counting problems for *full* LTL by placing the problems into counting complexity classes. These classes are based on counting accepting runs of nondeterministic Turing machines. In his seminal paper on the complexity of computing the permanent [20], Valiant introduced the class #P of counting problems associated with counting accepting runs of nondeterministic polynomial time Turing machines: a function $f\colon \Sigma^* \to \mathbb{N}$ is in #P if there is a nondeterministic polynomial time Turing machine $\mathcal{M}$ such that $f(w)$ is equal to the number of accepting runs of $\mathcal{M}$ on w. Furthermore, for a class $\mathcal{C}$ of decision problems, he defined[1] $\#_o\mathcal{C}$ to be the class of counting problems induced by counting accepting runs of a nondeterministic polynomial time Turing machine with an oracle from $\mathcal{C}$.

A nondeterministic polynomial time Turing machine $\mathcal{M}$ (with or without oracle) has at most $\mathcal{O}(2^{p(n)})$ different runs on inputs of length n for some polynomial p. This means that there is an exponential upper bound on functions in #P and in $\#_o\mathcal{C}$ for every $\mathcal{C}$. However, an LTL tautology has exponentially many word-models of length k and more than doubly-exponentially many tree-models of height k. This means, that no function in any of the counting classes defined above can capture the counting problems for LTL.

To overcome this, we consider counting classes obtained by lifting the restriction on considering only nondeterministic polynomial time (oracle) machines: a function $f\colon \Sigma^* \to \mathbb{N}$ is in #PSPACE, if there is a nondeterministic polynomial *space* Turing machine $\mathcal{M}$ such that $f(w)$ is equal to the number of accepting runs of $\mathcal{M}$ on w. The classes #EXPTIME, #EXPSPACE, and #2EXPTIME are defined analogously[2]. Some of these classes appeared in the literature, e.g., #PSPACE was shown to be equal to FPSPACE [11] (if the output is encoded in binary). Also, computing a specific entry of a matrix power A^n is in #PSPACE, if A is represented succinctly and n in binary [14], and counting self-avoiding walks in succinctly represented hypercubes is complete for #EXPTIME [12] under right-bit-shift reductions.

[1] Valiant originally used the notation $\#\mathcal{C}$, but we added the subscript to distinguish the oracle-based classes from the classes introduced below.

[2] Following the "satanic" [9] tradition of naming counting classes, we drop the N (standing for nondeterministic) in the names of the classes, just as it is done for #P.

We place the LTL model counting problems in these classes. Unsurprisingly, the encoding of the bound k is crucial: for unary bounds, we show counting word-models to be #P-complete and show counting tree-models to be #EXPTIME-complete. For binary bounds, the word-model counting problem is #PSPACE-complete and counting tree-models is #EXPSPACE-hard and in #2EXPTIME. The upper bounds hold for full LTL while the formulas for the lower bounds define safety properties (using only the temporal operators next and release). Thus, the lower bounds already hold for the fragment considered in [8].

The algorithms we present to prove the upper bounds are not practical since they are based on guessing a word (tree) and then model checking it. Hence, a deterministic variant of these algorithms would enumerate all words (trees) of length (height) k and then run a model checking algorithm on them. In particular, the running time of the algorithms is exponential (or worse) in the bound k, which is in stark contrast to the practical algorithms [8]. Our lower bounds are reductions from the problem of counting accepting runs of a Turing machine. For the word counting problem, the reductions are slight strengthenings of the reduction proving PSPACE-hardness of the LTL model checking problem [18]. However, the reductions in the tree case are more involved (and to the best of our knowledge new), since we have to deal with exponential time respectively exponential space Turing machines. The main technical difficulties are to encode runs of exponential length and with configurations of exponential size into tree-models of "small" LTL formulas and to ensure that there is a one-to-one correspondence between accepting runs and models of the constructed formula.

All proofs omitted due to space restrictions can be found in the full version [19].

2 Preliminaries

We represent models as *labeled transition systems.* For a given finite set Υ of directions and a finite set Σ of labels, a Σ-labeled Υ-transition system is a tuple $\mathcal{S} = (S, s_0, \tau, o)$, consisting of a finite set of states S, an initial state $s_0 \in S$, a transition function $\tau \colon S \times \Upsilon \to S$, and a labeling function $o \colon S \to \Sigma$. A *path* in $\mathcal{S}$ is a sequence $\pi \colon \mathbb{N} \to S \times \Upsilon$ of states and directions that follows the transition function, i.e., for all $i \in \mathbb{N}$ if $\pi(i) = (s_i, e_i)$ and $\pi(i+1) = (s_{i+1}, e_{i+1})$, then $s_{i+1} = \tau(s_i, e_i)$. We call the path initial if it starts with the initial state: $\pi(0) = (s_0, e)$ for some $e \in \Upsilon$.

We use Linear-time Temporal Logic (LTL) [16], with the usual temporal operators Next $\bigcirc$, Until $\mathcal{U}$, Release $\mathcal{V}$, and the derived operators Eventually $\diamondsuit$ and Globally $\square$. We use $\bigcirc^i$ to refer to i nested next operators. LTL formulas are defined over a set of atomic propositions $AP = I \cup O$, which is partitioned into a set I of input propositions and a set O of output propositions. We denote the satisfaction of an LTL formula φ by an infinite sequence $\sigma \colon \mathbb{N} \to 2^{AP}$ of valuations of the atomic propositions by $\sigma \models \varphi$. A 2^O-labeled 2^I-transition system $\mathcal{S} = (S, s_0, \tau, o)$ satisfies φ, if for every initial path π the sequence $\sigma_\pi \colon i \mapsto o(\pi(i))$, where $o(s, e) = (o(s) \cup e)$, satisfies φ. Then $\mathcal{S}$ is a model of φ.

A *k-word-model* of an LTL formula φ over AP is a pair (u, v) of finite words over 2^{AP} such that $|u.v| = k$ and $u.v^\omega \models \varphi$. We call u the prefix and v the period of (u, v). Note that an ultimately periodic word might be induced by more than one k-word-model, i.e., $\{a\}^\omega$ is induced by the 2-word-models $(\{a\}, \{a\})$ and $(\varepsilon, \{a\}\{a\})$.

A *k-tree-model* of an LTL formula φ over $AP = I \cup O$ is a 2^O-labeled 2^I-transition system that forms a tree (whose root is the initial state) of height k with added back-edges from the leaves (for each leaf and direction, there is an edge to a state on the branch leading to the leaf) that satisfies φ. Figure 1 shows a tree-model of height one.

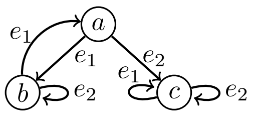


Figure 1 A tree-model.

Fix $AP = I \cup O$. For a formula φ and $k \in \mathbb{N}$, the *k-word* (*k-tree*) *counting problem* asks to compute the number of k-word-models (k-tree-models up to isomorphism) of φ over AP.

3 Counting Complexity Classes

We use nondeterministic Turing machines with or without oracle access to define counting complexity classes, which we assume (without loss of generality) to terminate on every input. For background on (oracle) Turing machines and counting complexity we refer to [1].

A function $f\colon \Sigma^* \to \mathbb{N}$ is in the class #P [20] if there is a nondeterministic polynomial time Turing machine $\mathcal{M}$ such that $f(w)$ is equal to the number of accepting runs of $\mathcal{M}$ on w. Similarly, for a given complexity class $\mathcal{C}$ of decision problems, a function f is in $\#_o\mathcal{C}$ [20, 9] if there is a nondeterministic polynomial time oracle Turing machine $\mathcal{M}$ with oracle in $\mathcal{C}$ such that $f(w)$ is equal to the number of accepting runs of $\mathcal{M}$ on w. As a nondeterministic polynomial time Turing machine $\mathcal{M}$ (with or without oracle) has at most $\mathcal{O}(2^{p(n)})$ runs on inputs of length n for some polynomial p (that only depends on $\mathcal{M}$), we obtain an exponential upper bound on functions in #P and $\#_o\mathcal{C}$ for every $\mathcal{C}$, which explains the need for larger counting classes to characterize the model counting problems for LTL.

A function $f\colon \Sigma^* \to \mathbb{N}$ is in #PSPACE, if there is a nondeterministic polynomial *space* Turing machine $\mathcal{M}$ such that $f(w)$ is equal to the number of accepting runs of $\mathcal{M}$ on w. #EXPTIME, #EXPSPACE, and #2EXPTIME are defined by counting accepting runs of nondeterministic exponential time, exponential space, and doubly-exponential time machines.

▶ **Proposition 1.**
1. $\#\mathrm{P} \subseteq \#_o\textsc{Pspace} \subseteq \#_o\textsc{Exptime} \subseteq \#_o\textsc{NExptime} \subseteq \#_o\textsc{Expspace} \subseteq \#_o2\textsc{Exptime}$.
2. $\#\textsc{Pspace} \subseteq \#\textsc{Exptime} \subsetneq \#\textsc{Expspace} \subseteq \#2\textsc{Exptime}$.
3. $f \in \#\textsc{Exptime}$ *implies* $f(w) \in \mathcal{O}(2^{2^{p(|w|)}})$ *for a polynomial* p.
4. $f \in \#2\textsc{Exptime}$ *implies* $f(w) \in \mathcal{O}(2^{2^{2^{p(|w|)}}})$ *for a polynomial* p.
5. $w \mapsto 2^{2^{|w|}}$ *is in* #PSPACE
6. $w \mapsto 2^{2^{2^{|w|}}}$ *is in* #EXPSPACE.

We continue by relating the oracle-based and the generalized classes introduced above.

▶ **Lemma 1.** $\#_oC \subsetneq \#C$ *for* $C \in \{\textsc{Pspace}, \textsc{Exptime}, \textsc{Expspace}, 2\textsc{Exptime}\}$.

Proof. We show $\#_o\textsc{Pspace} \subsetneq \#\textsc{Pspace}$, the other claims are proven analogously. Let $f \in \#_o\textsc{Pspace}$, i.e., there is a nondeterministic polynomial time Turing machine $\mathcal{M}$ with oracle $A \in \textsc{Pspace}$ such that $f(w)$ is equal to the number of accepting runs of $\mathcal{M}$ on w. Note that all oracle queries are polynomially-sized in the length $|w|$ of the input to $\mathcal{M}$, since $\mathcal{M}$ is polynomially time-bounded. Hence, in nondeterministic polynomial space one can simulate $\mathcal{M}$ and evaluate the oracle calls explicitly by running a deterministic machine deciding A in polynomial space. Since the oracle queries are evaluated deterministically, the simulation has as many accepting runs as $\mathcal{M}$ has. Thus, $f \in \#\textsc{Pspace}$.

Now, consider the function $|w| \mapsto 2^{2^{|w|}}$, which is in #PSPACE, but not in $\#_o\textsc{Pspace}$. ◀

We use parsimonious reductions to define hardness and completeness, i.e., the most restrictive notion of reduction for counting problems. A counting problem f is #P-hard, if for every $f' \in \#\mathrm{P}$ there is a polynomial time computable function r such that $f'(x) = f(r(x))$ for all inputs x. In particular, if f' is induced by counting the accepting runs of $\mathcal{M}$, then r depends on $\mathcal{M}$ (and possibly on its time-bound $p(n)$). Furthermore, f is #P-complete, if f is #P-hard and $f \in \#\mathrm{P}$. Hardness and completeness for the other classes are defined analogously.

4 Counting Word-Models

In this section, we provide matching lower and upper bounds for the complexity of counting k-word-models of an LTL specification.

Our hardness proofs are based on constructing an LTL formula $\varphi_{\mathcal{M}}^{w}$ for a given Turing machine $\mathcal{M}$ and an input w that encodes the accepting runs of $\mathcal{M}$ on w. Constructing such an LTL formula is straightforward and can be done in polynomial time for Turing machines with polynomially-sized configurations [18]. However, the challenge is to construct $\varphi_{\mathcal{M}}^{w}$ such that the number of accepting runs on w is equal to the number of k-word-models of $\varphi_{\mathcal{M}}^{w}$ for a fixed bound k. To this end, we have to enforce that each accepting run is represented by a unique k-word-model, i.e., by a unique prefix and period of total length k. We choose k such that a run on w of maximal length can be encoded in $k-1$ symbols and define $\varphi_{\mathcal{M}}^{w}$ such that it has only k-word-models whose period has length one. If a run of $\mathcal{M}$ is shorter than the maximal-length run we repeat the final configuration until reaching the maximal length, which is achieved by accompanying the configurations in the encoding with consecutive id's.

For the upper bounds we show that there are appropriate nondeterministic Turing machines that guess an ultimately-periodic word and model check it against φ, i.e., the number of accepting runs on k and φ is equal to the number of k-word-models of φ.

The Case of Unary Encodings. We show that counting word-models for unary bounds is #P-complete.

▶ **Theorem 2.** *The following problem is* #P*-complete: Given an LTL formula φ and a bound k (in unary), how many k-word-models does φ have?*

Proof. We start with the hardness proof. Let $\mathcal{M} = (Q, q_\iota, Q_F, \Sigma, \delta)$ be a one-tape nondeterministic polynomial time Turing machine, where Q is the set of states, q_ι is the initial state, Q_F is the set of accepting states, Σ is the alphabet, and $\delta\colon (Q \setminus Q_F) \times \Sigma \to 2^{Q\times\Sigma\times\{-1,1\}}$ is the transition function, where -1 and 1 encode the directions of the head. Note that the accepting states are terminal and that $\mathcal{M}$ rejects by terminating in a nonaccepting state. Let $\mathcal{M}$ be $p(n)$-time bounded for some polynomial p, and let $w = w_0 \cdots w_{n-1}$ be an input to $\mathcal{M}$. We construct an LTL formula $\varphi_{\mathcal{M}}^{w}$ and define a bound k, both polynomial in $|w|$ and $|\mathcal{M}|$, such that the number of accepting runs of $\mathcal{M}$ on w is equal to the number of k-word-models of $\varphi_{\mathcal{M}}^{w}$.

A run of $\mathcal{M}$ on w is encoded by a finite alternating sequence of id's id_i and configurations c_i that is followed by an infinite repetition of a dummy symbol:

$$\$ \; \mathrm{id}_0 \; \# \; c_0 \; \$ \; \mathrm{id}_1 \; \# \; c_1 \; \$ \; \mathrm{id}_2 \; \# \; c_2 \; \$ \; \cdots \; \$ \; \mathrm{id}_{p(n)} \; \# \; c_{p(n)} \; (\bot)^\omega \tag{1}$$

Note that the period of the word-model is of the form $\bot^\ell$ for some $\ell > 0$. We will define k such that maximal-length runs of $\mathcal{M}$ on w can be encoded in the prefix, and such that the only possible period has length one by ensuring that exactly $p(n)$ configurations are encoded (by repeating the final configuration if necessary). This ensures that an accepting run is encoded by exactly one k-word-model.

Let $l_r = p(n)$ be the maximal length of a run of $\mathcal{M}$ on w. The size of a configuration of $\mathcal{M}$ on w is also bounded by l_r. For the id's we use an encoding of a binary counter with $l_c = \lceil \log l_r \rceil$ many bits. Let $AP = (Q \cup \Sigma) \uplus \{b_1, \ldots, b_{l_c}, \$, \#, \bot\}$ be the set of atomic propositions. The atomic propositions in $Q \cup \Sigma$ are used to encode the configuration of $\mathcal{M}$ by encoding the tape contents, the state of the machine, and the head position. The atomic propositions $b_1, \ldots, b_{l_c}$ represent the bit values of an id. The symbols \$ and # are used as separators between id's and configurations, and $\perp$ is a dummy symbol for the model's period. The distance between two \$ symbols and also between two # symbols in the encoding is given by $d = l_r + 3$ (see (1)). Then, $\varphi^w_{\mathcal{M}}$ is the conjunction of the following formulas:

- *Id* encodes the id's of the configurations. It uses a formula $Inc(b_1, \ldots, b_{l_c}, d)$ that asserts that the number encoded by the bits b_j after d steps is obtained by incrementing the number encoded at the current position. This formula will be reused in the tree case.
- *Init* asserts that the run of $\mathcal{M}$ starts with the initial configuration.
- *Accept* asserts that the run must reach an accepting configuration.
- *Config* declares the consistency of two successive configurations with the transition relation of $\mathcal{M}$. Here, we use d many next operators to relate the encoding of the two configurations.
- *Repeat* asserts that the encoding of an accepting configuration is repeated until the maximal id is reached
- *Loop* defines the period of the word-model, which may only contain $\perp$.

All these properties can be expressed with polynomially-sized formulas, which can be found in the full version [19]. Furthermore, we need a formula to specify technical details: atomic propositions encoding the id's are not allowed to appear in the configurations and vice versa, symbols such as \$ and # only to appear as separators, each separator appears $p(n)$ times every d positions, configuration encodings are represented by singleton sets of letters in Σ with the exception of one set that contains a symbol from Q to determine the head position and the state of $\mathcal{M}$, etc.

For $k = l_r \cdot (l_r + 3) + 1$, each accepting run of $\mathcal{M}$ on w corresponds to exactly one k-word-model of $\varphi^w_{\mathcal{M}}$ that encodes the run in its prefix. Thus, the number of k-word-models is equal to the number of accepting runs of $\mathcal{M}$ on w. The formula $\varphi^w_{\mathcal{M}}$ can be obtained in polynomial time in $|w| + |\mathcal{M}|$, and k (thus also its unary encoding) is polynomial in $|w|$.

To show that the problem is in #P we define a nondeterministic polynomial time Turing machine $\mathcal{M}$ as follows. $\mathcal{M}$ guesses a prefix u and a period v of an ultimately periodic word $u.v^\omega$ with $|u.v| = k$, and checks deterministically in polynomial time [10], whether $u.v^\omega$ satisfies φ. Hence, for each k-word-model (u, v) of φ there is exactly one accepting run of $\mathcal{M}$. Thus, counting the k-word-models of φ can be done by counting the accepting runs of $\mathcal{M}$ on the input (k, φ). ◀

The Case of Binary Encodings. Now, we consider the word counting problem for binary bounds. As the input is more compact, we have to deal with a larger complexity class.

▶ **Theorem 3.** *The following problem is* #PSPACE*-complete: Given an LTL formula φ and a bound k (in binary), how many k-word-models does φ have?*

Proof. The hardness proof is similar to the one for Theorem 2: for a nondeterministic polynomial space Turing machine $\mathcal{M}$ bounded by a polynomial $p(n)$ and an input word w we can define a formula $\varphi^w_{\mathcal{M}}$ in the same way as in Theorem 2. The reason lies in that the size of configurations remains polynomial and the exponential number of configurations in a run can still be counted with a binary counter of polynomial size, i.e., we only have to use more bits b_j to encode the id's. Furthermore, we have to choose $k = 2^{p'(n)}(p(n) + 3) + 1$ which can still be encoded using polynomially many bits. Here, $p'(n)$ is a polynomial (which only depends on $\mathcal{M}$) such that $\mathcal{M}$ terminates in at most $2^{p'(n)}$ steps on inputs of length n.

For the proof of the upper bound we cannot just guess a k-model in polynomial space as in Theorem 2, since the bound k is encoded in binary. Instead, we guess and verify the model on-the-fly relying on standard techniques for LTL model checking.

Formally, we construct a nondeterministic polynomial space Turing machine $\mathcal{M}$ which guesses a k-word-model (u, v) by guessing $u\$v = w(0) \cdots w(i-1)\$w(i) \cdots w(k-1)$ symbol by symbol in a backwards fashion. Here, \$ is a fresh symbol to denote the beginning of the period. To meet the space requirement, $\mathcal{M}$ only stores the currently guessed symbol $w(j)$, discards previously guessed symbols, and uses a binary counter to guess exactly k symbols.

To verify whether $u.v^\omega$ satisfies φ, $\mathcal{M}$ also creates for every j in the range $0 \leq j < k$ a set C_j of subformulas of φ with the intention of C_j containing exactly the subformulas which are satisfied in position j of $u.v^\omega$. Due to space-requirements, $\mathcal{M}$ only stores the set C_{k-1} as well as the sets C_j and C_{j+1}, if $w(j)$ is the currently guessed symbol. The set C_{k-1} is guessed by $\mathcal{M}$ and the sets C_j for $j < k-1$ are uniquely determined by the following rules:

- The membership of atomic propositions in C_j is determined by $w(j)$, i.e., $C_j \cap AP = w(j)$.
- Conjunctions, disjunctions, and negations can be checked locally for consistency, e.g., $\neg\psi \in C_j$ if and only if $\psi \notin C_j$.
- $\bigcirc$-formulas are propagated backwards using the following equivalence: $\bigcirc\psi \in C_j$ if and only if $\psi \in C_{j+1}$ (recall that $\mathcal{M}$ stores C_j and C_{j+1}).
- $\mathcal{U}$-formulas are propagated backwards using the following equivalence: $\psi_0\mathcal{U}\psi_1 \in C_j$ if and only if $\psi_1 \in C_j$ or $\psi_0 \in C_j$ and $\psi_0\mathcal{U}\psi_1 \in C_{j+1}$.
- $\mathcal{V}$-formulas can be rewritten into $\mathcal{U}$-formulas.

Once $\mathcal{M}$ has guessed the complete period $v = w(i) \cdots w(k-1)$ it also checks that the guess of C_{k-1} is correct (recall that C_{k-1} is not discarded), which is the case if the following two requirements are met:

- For every subformula $\bigcirc\psi$ we have $\bigcirc\psi \in C_{k-1}$ if and only if $\psi \in C_i$.
- For every subformula $\psi_0\mathcal{U}\psi_1$ we have $\psi_0\mathcal{U}\psi_1 \in C_{k-1}$ if and only if $\psi_1 \in C_{k-1}$ or $\psi_0 \in C_{k-1}$ and $\psi_0\mathcal{U}\psi_1 \in C_i$. Furthermore, we have to require that $\psi_0\mathcal{U}\psi_1 \in C_j$ for some j in the range $i \leq j < k$ implies $\psi_1 \in C_{j'}$ for some j' in the range $i \leq j' < k$. The latter condition can be checked on-the-fly while computing the C_j's.

A straightforward structural induction over the construction of φ shows that we have $\psi \in C_j$ if and only if $w(j)w(j+1) \cdots w(k-1)v^\omega \models \psi$ for every subformula ψ of φ. Hence, $u.v^\omega$ is a model of φ if and only if $\varphi \in C_0$. Thus, $\mathcal{M}$ accepts if this is the case. ◀

5 Counting Tree-Models

In this section, we consider the tree counting problem for unary and binary bounds. There are at least doubly-exponentially many trees of height k. Hence, if k is encoded in binary, there are at least triply-exponentially many (in the size of the encoding of k) k-tree-models of a tautology. In order to capture these cardinalities using counting classes, we have to consider machines with that many runs, i.e., exponential time and exponential space machines.

In our hardness proofs, we again construct formulas $\varphi_{\mathcal{M}}^w$ that encode accepting runs of $\mathcal{M}$ on w in trees. We choose binary trees, i.e., we consider a singleton set I of input propositions. Recall that the power set of I is used to (deterministically) label the edges in the tree. In the following, we identify the two elements of 2^I with the directions `left` and `right`. Note that we have to formalize the structure of our models and have to encode the runs of the machines using LTL. The semantics require a formula to be satisfied on all paths, which requires us to write conditional formulas of the form "if the path has a certain form, then some property is satisfied". We use two types of formulas: the ones of the first type describe the structure of the tree (e.g., it is complete and the targets of the back-edges) while the ones of the second type encode the actual run relying on this structure. The formulas of type one often assign addresses to nodes (sequences of bits that uniquely identify a leaf).

In the word case, we encoded runs of Turing machines whose configurations are of polynomial length. Hence, the distance between encodings of a tape cell in two successive configurations could be covered by a polynomial number of next-operators. Here, configurations are of exponential size. Thus, the challenge is to encode a run in a tree-model such that properties of two successive configurations can still be encoded by an LTL formula of polynomial size. We present two such encodings, one for unary and one for binary bounds.

For the upper bounds we show that there are appropriate nondeterministic machines that guess a finite tree with back-edges and model check it deterministically against φ, i.e., the number of accepting runs on k and φ is equal to the number of k-tree-models of φ.

The Case of Unary Encodings. First, we consider tree-model counting for unary bounds.

▶ **Theorem 4.** *The following problem is* #EXPTIME*-complete: Given an LTL formula φ and a bound k (in unary), how many k-tree-models does φ have?*

Proof. We start with the hardness proof. Let $\mathcal{M} = (Q, q_\iota, Q_F, \Sigma, \delta)$ be a one-tape nondeterministic exponential time Turing machine. Let $\mathcal{M}$ be $2^{p(n)}$-time bounded for a polynomial p and let $w = w_0 \cdots w_{n-1}$ be an input to $\mathcal{M}$. We construct an LTL formula $\varphi^w_\mathcal{M}$ and define a bound k, both polynomial in $|w|$ and $|\mathcal{M}|$, such that the number of accepting runs of $\mathcal{M}$ on w is equal to the number of k-tree-models of $\varphi^w_\mathcal{M}$.

A run of $\mathcal{M}$ is encoded in the leaves of a binary tree-model. Let $l_r = 2^{p(n)}$ be the maximal length of a run of $\mathcal{M}$ on w, which also bounds the size of a configuration. We choose $k = 2p(n)$ to be the height of our tree-models. By using a formula labeling each of the first k levels of the tree by a unique proposition we enforce that every model of height k is complete. Thus, it has l_r^2 many leaves, enough to encode a run of maximal length. Figure 2 shows the structure of our tree-model.

Each configuration in the run is encoded in the leaves of a subtree of height $p(n)$, referred to as a *lower*-tree (depicted by the light gray trees). The lower-trees are uniquely determined by a leaf of the *upper*-tree (depicted in dark gray), which is the root of the lower-tree. By giving the leaves of the upper-tree id's, we also obtain unique id's for each of the lower-trees. These id's are used to enumerate the configurations of the run, i.e., two neighboring lower-trees encode two successive configurations of the run. The id's can be determined by a binary counter with polynomially many bits. We also provide each leaf in a lower-tree with a unique id within this lower-tree. This is used to compare the contents of a tape cell in two successive configurations by comparing the labels of leaves with the same leaf id in two successive lower-trees. Thus, every leaf stores the id encoding of the configuration it is part of and the number of the cell it encodes.

Recall that in a tree-model each leaf has a back-edge for every direction. For the direction `left` we require a transition to the root of the upper-tree, and for `right` a transition to the root of the own lower-tree. This enables us to compare two leaves in a lower-tree, or two leaves with the same id in two different lower-trees, with polynomially large formulas.

The following formulas define the structure of our tree-models as explained above and also provide the nodes of the tree with correct id's. We begin with $\mathit{Addr}(\texttt{root}, a_1, \ldots, a_d)$ which specifies a unique id for each leaf of a complete binary tree of height d using bits $a_1, \ldots, a_d$, and provides the root of the tree with a label `root`. The id of a node depends on the sequence of `left` and `right` edges on the path from the root to this node, which is encoded in the bits $a_1, \ldots, a_d$:

$$\mathit{Addr}(\texttt{root}, a_1, \ldots, a_d) = \texttt{root} \wedge \bigwedge_{i=0}^{d-1} \left(\bigcirc^i (\texttt{left} \rightarrow \bigcirc^{d-i} \neg a_{i+1}) \wedge \bigcirc^i (\texttt{right} \rightarrow \bigcirc^{d-i} a_{i+1}) \right) .$$

We use the formula $\mathit{Addr}(\texttt{upper}, u_1, \ldots, u_{p(n)})$ to address the upper-tree. This gives each lower-tree a unique id via the id of its root. We also supply each node in a lower-tree with the id of its root in the upper-tree: $\bigwedge_{p(n) \leq i < k} \bigcirc^i (\bigwedge_{j=1}^{p(n)} (u_j \leftrightarrow \bigcirc u_j))$. Furthermore, we use the formula $\bigcirc^{p(n)} \mathit{Addr}(\texttt{lower}, l_1, \ldots, l_{p(n)})$ to assign every leaf in a lower-tree a unique id within its lower-tree which essentially encodes the number of the tape cell it encodes. The next two formulas define the back-edges of the lower-trees. From each leaf, the `left` transition leads back to the root of the upper-tree (recall that back-edges lead from a leaf to an ancestor), i.e., $\bigcirc^k(\texttt{left} \rightarrow \bigcirc \texttt{upper})$, and the `right` transition to the root of the lower-tree, i.e., $\bigcirc^k(\texttt{right} \rightarrow \bigcirc \texttt{lower})$. After setting up the structure of the trees, it remains to show how we encode a run in the leaves. We proceed with the same scheme as in the word case, and use the formula $\Delta_h(a_1, \ldots, a_m)$ which is satisfied, if and only if the bits $a_1, \ldots, a_m$ encode the number $h < 2^m$.

- The formula *Init* encodes the initial configuration in the lower-tree with id 0.

$$\bigcirc^k \Big[\Delta_0(u_1, \ldots, u_{p(n)}) \rightarrow \Big((\Delta_0(l_1, \ldots, l_{p(n)}) \rightarrow q_\iota) \wedge \bigwedge_{0 \leq j < n} (\Delta_j(l_1, \ldots, l_{p(n)}) \rightarrow w_j) \\ \wedge ((\bigwedge_{0 \leq j < n} \neg\Delta_j(l_1, \ldots, l_{p(n)})) \rightarrow \sqcup)\Big)\Big] .$$

- The formula *Accept* checks whether the rightmost lower-tree encodes an accepting configuration: $\bigcirc^k((\Delta_{l_r}(u_1, \ldots, u_{p(n)}) \wedge \bigvee_{q \in Q} q) \rightarrow \bigvee_{q \in Q_F} q)$.
- The formulas $\mathit{config}_{q,\alpha}$ and config_α for states q and symbols α encode the transition relation. For a leaf with labels q and α (leaf 1 in Figure 2) and a transition $(q, \alpha, q', \beta, \texttt{dir})$, we have to check three leaves in the next lower-tree, namely, the leaf with the same id (leaf 2) has to be labeled with β, and depending on `dir` either the successor leaf (leaf 3) or the predecessor leaf (leaf 4) has to be labeled with q'. The premise of the following formula only holds for paths that visit these leaves in the order given above, i.e., paths that lead to a leaf in a lower-tree, loop back to the root of the full tree and then lead to the same leaf id in the successor lower-tree (this takes $k+1$ edges), loop back to the root of this lower-tree and visit the leaf to the right (this takes $p(n)+1$ edges), back to the root of this lower-tree again and then to the leaf to the left (this takes $p(n)+1$ edges). To specify such a path, we use the formula *Inc* to reach the successor leaf and a dual formula called *Dec* to reach the predecessor leaf. This formula implements a decrement of a nonzero counter. Note that we have to require the paths to visit the successor and predecessor leaf in the next lower-tree, i.e., we have to check the bits u_j to reach the next lower-tree and the bits l_j to reach the leaves. Thus, $\mathit{config}_{q,\alpha}$ for $q \in Q \setminus Q_F$ is given by:

$$\bigcirc^k \Big[\, q \wedge \alpha \wedge \mathit{Inc}(u_1, \ldots, u_{p(n)}, k+1) \wedge \bigwedge_{i=1}^{p(n)} l_i \leftrightarrow \bigcirc^{k+1} l_i) \\ \wedge \mathit{Inc}(u_1, \ldots, u_{p(n)}, k+p(n)+2) \wedge \mathit{Inc}(l_1, \ldots, l_{p(n)}, k+p(n)+2) \\ \wedge \mathit{Inc}(u_1, \ldots, u_{p(n)}, k+2p(n)+3)) \wedge \mathit{Dec}(l_1, \ldots, l_{p(n)}, k+2p(n)+3) \\ \rightarrow \bigvee_{(q', \beta,\ \texttt{dir}) \in \delta(q,\alpha)} (\bigcirc^{k+1} \beta \wedge \bigcirc^{(k+1)+c_{\texttt{dir}}(p(n)+1)} q') \,\Big] .$$

Here, we have $c_{\texttt{dir}} = 1$, if $\texttt{dir} = 1$, and $c_{\texttt{dir}} = 2$, if $\texttt{dir} = -1$.
The formula config_α determines the relation between the other tape cells' contents, namely where the head is not pointing to:

$$\bigcirc^k(\bigvee_{i=1}^{p(n)} \neg u_i \wedge (\bigwedge_{q \in Q \setminus Q_F} \neg q) \wedge \alpha \wedge \mathit{Inc}(u_1, \ldots, u_{p(n)}, k+1) \wedge (\bigwedge_{i=1}^{p(n)} l_i \leftrightarrow \bigcirc^{k+1} l_i) \rightarrow \bigcirc^{k+1} \alpha) .$$

- The formula *Repeat* repeats accepting states in the next lower-tree, if the id of the current lower-tree is not maximal. The repetition of the letters is being taken care of by config_α.

$$\bigcirc^k \big[\big(\bigvee_{i=1}^{p(n)} \neg u_i \wedge \mathit{Inc}(u_1, \ldots, u_{p(n)}, k+1) \wedge \bigwedge_{i=1}^{p(n)} (l_i \leftrightarrow \bigcirc^{k+1} l_i) \big) \rightarrow \big(\bigwedge_{q_f \in Q_F} q_f \rightarrow \bigcirc^{k+1} q_f \big) \big] .$$

Similar to the word case we need some additional formulas to prevent atomic propositions of configurations to appear elsewhere in the tree to guarantee the one-to-one relation between runs and tree-models. For example to prevent a state label from appearing twice in a configuration we use a formula that asserts that from a leaf in which a state is encoded, no other leaf with a state label is reachable within $p(n) + 1$ steps, i.e., in the same lower-tree. This ensures that every configuration has exactly one state.

To show that the problem is in #EXPTIME we define a nondeterministic exponential time Turing machine $\mathcal{M}$ as follows. $\mathcal{M}$ guesses a tree of height k (which is of exponential size) and checks whether it satisfies φ using the classical model checking algorithm: $\mathcal{M}$ constructs the Büchi automaton recognizing the language of $\neg\varphi$ and checks whether the product of the tree and the automaton has an empty language. The automaton and the product are of exponential size and the emptiness check can be performed in deterministic polynomial time (in the size of the product). Hence, $\mathcal{M}$ runs in exponential time in k and the size of φ. For each k-tree-model of φ, there is exactly one accepting run in $\mathcal{M}$. Thus, counting the k-tree-models of φ can be done by counting the accepting runs of $\mathcal{M}$ on the input (k, φ). ◀

The Case of Binary Encodings. In this section, we consider tree-model counting for binary bounds. Since the bound is encoded compactly, the trees we work with have exponential height and therefore doubly-exponential size. Unfortunately, our upper and lower bounds do not match (see the discussion in the next section).

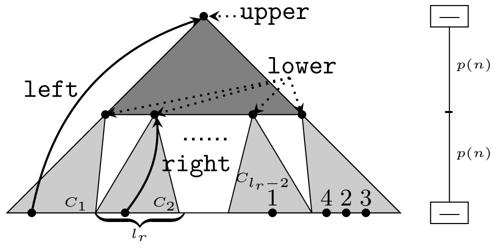


Figure 2 Encoding an exponentially long run in a tree-model of polynomial height. The configurations are encoded in the lower-trees (light gray subtrees).

▶ **Theorem 5.** *The following problem is #EXPSPACE-hard and in #2EXPTIME: Given an LTL formula φ and a bound k (in binary), how many k-tree-models does φ have?*

Proof. Let $\mathcal{M} = (Q, q_\iota, Q_F, \Sigma, \delta)$ be a one-tape nondeterministic exponential space Turing machine and let $w = w_0 \cdots w_{n-1}$ be an input to $\mathcal{M}$. Furthermore, let $l_c = 2^{p(n)} - 2$ be the maximal configuration length (for some polynomial p) and let $l_r = 2^{2^{p'(n)}}$ be the maximal length of a run of $\mathcal{M}$ on w (p' is a polynomial which only depends on $\mathcal{M}$).

We choose $k = m \cdot 2^{p'(n)}$ to be the height of our tree-models, where m is the smallest power of two greater than $p(n)$. Figure 3 shows the main structure of our tree-models. We use nonbalanced binary trees that are composed of trees of height m. We refer to the latter trees as the *inner*-trees. The outermost leaves of an inner-tree are inner nodes and the others are leaves in the tree-model. Hence, each inner-tree has two children, which are again inner-trees rooted at the leftmost respectively the rightmost leaf.

In each inner-tree, we will encode a configuration in a similar way as in the unary case (Theorem 4), namely in the leaves (except the two leaves serving as roots for other inner trees, which explains the -2 in the definition of l_c). We encode the configurations of a run

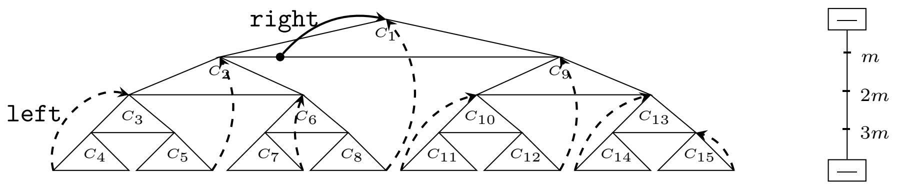


Figure 3 Tree-model with DFS structure.

in the tree-model such that we traverse the inner-trees in a depth-first search manner (DFS). In Figure 3, we can see how a run of 16 configurations can be encoded in a tree-model with four layers of inner-trees. To encode the DFS structure, we label each root of an inner-tree with its *level* (the number of inner-tree ancestors) and with its so-called *right-child-depth*: the number of right-child-inner-trees visited since the last left child to reach this tree (e.g., this value is 0 for the left children C_1, C_2, C_3, C_7; it is 1 for C_6 and 3 for C_{15}). This will help to determine the next inner-tree in line in the DFS structure. We need a polynomial number of bits to encode these addresses. With the `right` transition we allow the leaves of an inner-tree to reach its root and we use `left` in the inner-tree of maximal level to reach the parent of the next inner-tree in DFS order. In this way, the distance between the encoding of a tape cell in two successive configurations is polynomial.

As the distance between an inner-tree and its successor is polynomial, the formulas for encoding the run in the tree-model adapt the ideas of the formulas in the unary case with slight modifications that deal with the DFS order of inner-trees. A detailed description of the construction can be found in the full version [19].

The upper bound is proved using the same algorithm as in the proof of Theorem 4. ◀

6 Discussion

We investigated the complexity of the model counting problem for specifications in Linear-time temporal logic. The word-model counting problems are #P-complete (for unary bounds) respectively #PSPACE-complete (for binary bounds) while the tree-model counting problems are #EXPTIME-complete respectively #EXPSPACE-hard and in #2EXPTIME, i.e., the exact complexity of the tree-model counting problem for binary bounds is open.

The problem we face trying to lower the upper bound is that we cannot guess the complete tree-model in nondeterministic exponential space. To meet the space-requirements, we have to construct it step by step, as in the proof of the corresponding upper bound in the word case. However, the correctness of the on-the-fly model checking procedure described there relies on the fact that the model is an ultimately-periodic word. It is open whether the technique can be extended to tree-models. On the other hand, if we try to raise the lower bound, we have to encode doubly-exponential time Turing machines, which seems challenging using polynomially-sized LTL formulas.

To conclude, let us mention another variation of the model counting problem: counting arbitrary transition systems, where the bound k now refers to the size of the transition system. For unary bounds, the problem is #P-hard, which can be shown by strengthening Theorem 2, and in $\#_o$PSPACE, since LTL model checking is in PSPACE. For binary bounds, the construction presented in Theorem 4 yields #EXPTIME-hardness and the problem is in #EXPTIME, which can be shown by adapting the algorithm presented in the theorem.

Acknowledgments. We would like to thank Markus Lohrey and an anonymous reviewer for bringing Ladner's work on polynomial space counting [11] to our attention.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

2 Armin Biere. Bounded model checking. In *Handbook of Satisfiability*, pages 457–481. IOS Press, 2009.

3 Roderick Bloem, Hans-Jürgen Gamauf, Georg Hofferek, Bettina Könighofer, and Robert Könighofer. Synthesizing robust systems with RATSY. In Doron Peled and Sven Schewe, editors, *SYNT*, volume 84 of *EPTCS*, pages 47–53. Open Publishing Association, 2012.

4 Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *LNCS*, pages 652–657. Springer, 2012.

5 Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

6 Rüdiger Ehlers. Unbeast: Symbolic bounded synthesis. In *TACAS*, volume 6605 of *LNCS*, pages 272–275. Springer-Verlag, 2011.

7 Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5-6):519–539, 2013.

8 Bernd Finkbeiner and Hazem Torfah. Counting models of linear-time temporal logic. In Adrian Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *LATA*, volume 8370 of *LNCS*, pages 360–371. Springer, 2014.

9 Lane A. Hemaspaandra and Heribert Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.

10 Lars Kuhtz and Bernd Finkbeiner. LTL path checking is efficiently parallelizable. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *ICALP*, volume 5556 of *LNCS*, pages 235–246. Springer, 2009.

11 Richard E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.

12 Maciej Liśkiewicz, Mitsunori Ogihara, and Seinosuke Toda. The complexity of counting self-avoiding walks in subgraphs of two-dimensional grids and hypercubes. *Theor. Comput. Sci.*, 1–3(304):129–156, 2003.

13 Michael L. Littman, Stephen M. Majercik, and Toniann Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27:2001, 2000.

14 Markus Lohrey and Manfred Schmidt-Schauß. Processing succinct matrices and vectors. In Edward A. Hirsch, Sergei O. Kuznetsov, Jean-Éric Pin, and Nikolay K. Vereshchagin, editors, *CSR*, volume 8476 of *LNCS*, pages 245–258. Springer, 2014.

15 Daniel Morwood and Daniel Bryce. Evaluating temporal plans in incomplete domains. In Jörg Hoffmann and Bart Selman, editors, *AAAI*. AAAI Press, 2012.

16 Amir Pnueli. The temporal logic of programs. In *STOC*, SFCS'77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

17 A. Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.

18 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

19 Hazem Torfah and Martin Zimmermann. The complexity of counting models of linear-time temporal logic. *ArXiv e-prints*, abs/1408.5752, 2014.

20 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

Extending Temporal Logics with Data Variable Quantifications*

Fu Song[1] and Zhilin Wu[2,3]

1 **Shanghai Key Laboratory of Trustworthy Computing and National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, P. R. China**
fsong@sei.ecnu.edu.cn

2 **State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, P. R. China**
wuzl@ios.ac.cn

3 **LIAFA, Université Paris Diderot, France**

Abstract

Although data values are available in almost every computer system, reasoning about them is a challenging task due to the huge data size or even infinite data domains. Temporal logics are the well-known specification formalisms for reactive and concurrent systems. Various extensions of temporal logics have been proposed to reason about data values, mostly in the last decade. Among them, one natural idea is to extend temporal logics with variable quantifications ranging over an infinite data domain. In this paper, we focus on the variable extensions of two widely used temporal logics, Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). Grumberg, Kupferman and Sheinvald recently investigated the extension of LTL with variable quantifications. They defined the extension as formulas in the prenex normal form, that is, all the variable quantifications precede the LTL formulas. Our goal in this paper is to do a relatively complete investigation on this topic. For this purpose, we define the extensions of LTL and CTL by allowing arbitrary nestings of variable quantifications, Boolean and temporal operators (the resulting logics are called respectively variable-LTL, in brief VLTL, and variable-CTL, in brief VCTL), and identify the decidability frontiers of both the satisfiability and model checking problem. In particular, we obtain the following results: 1) Existential variable quantifiers or one single universal quantifier in the beginning already entails undecidability for the satisfiability problem of both VLTL and VCTL, 2) If only existential path quantifiers are used in VCTL, then the satisfiability problem is decidable, no matter which variable quantifiers are available. 3) For VLTL formulas with one single universal variable quantifier in the beginning, if the occurrences of the non-parameterized atomic propositions are guarded by the positive occurrences of the quantified variable, then its satisfiability problem becomes decidable. Based on these results of the satisfiability problem, we deduce the (un)decidability results of the model checking problem.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Temporal logics with variable quantifications, satisfiability and model checking, alternating register automata, data automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.253

* Fu Song is supported by the NSFC under Grant No. 61402179, STCSM Pujiang Talent Project under Grant No. 14PJ1403200, SHMEC&SHEDF ChenGuang Project under Grant No. 13CG21, the Open Project of Shanghai Key Laboratory of Trustworthy Computing under Grant No. 07dz22304201301, and Shanghai Knowledge Service Platform for Trustworthy Internet of Things under Grant No. ZF1213. Zhilin Wu is supported by the NSFC under Grant No. 61100062, 61272135, and 61472474, and partially supported by the visiting researcher program of China Scholarship Council.

1 Introduction

Context. Data values are ubiquitous in computer systems. To see just the tip of the iceberg, we have the following scenarios: data variables in sequential programs, process identifiers in concurrent parameterized systems where an unbounded number of processes interact with each other, records in relational databases, attributes of elements of XML documents or nodes of graph databases. On the other hand, reasoning about data values is a very challenging task. Either their sizes are huge, e.g. one single 4-byte integer variable in C programs may take values from $-2,147,483,64$ to $2,147,483,647$, or they are even from an infinite domain, e.g. process identifiers in parameterized systems.

Temporal logics are the widely used specification formalisms. They were originally introduced to formally specify and reason about the behaviors of concurrent and reactive systems. But later on, they also became popular in reasoning about the behaviors of sequential programs. Linear temporal logic (LTL) and computation tree logic (CTL) are the two most widely used temporal logics. Since temporal logics are targeted to specify the behaviors of finite state systems, various extensions of temporal logics have been proposed to deal with the data values, mostly in the last decade. Hodkinson et al. initiated the investigation of first-order extensions of temporal logics, that is, first-order logic over relational structures extended with temporal operators ([14, 13]). Their motivation is theoretical and they focused on the decidability issues. Vianu and his coauthors used the first-order extensions of LTL to specify and reason about the behaviors of database-driven systems ([22, 3]). Demri and Lazic extended LTL with freeze quantifiers which can store data values into registers and compare the data values with those stored in the registers ([6]). Schwentick et al., Demri et al. , and Decker et al. considered extensions of LTL with navigation mechanisms for data values or tuples over multi-attributed data words, that is, data words where each position carries multiple data values ([16, 5, 4]).

Another natural idea is to extend temporal logics with variable quantifications over an infinite data domain, which is our focus in this paper. There have been some work on this topic. Grumberg et al. investigated the extension of LTL with variable quantifications. They defined the extension as the formulas in prenex normal form, that is, all the variable quantifications are in the beginning and are followed by LTL formulas, and investigated the decidability of the satisfiability and the model checking problem over Kripke structures extended with data variables ([10, 11]). Figueira proposed an extension of LTL with freeze quantifiers of only one register, by quantifications over the set of data values occurring before or after the current position of data words ([8]).

Contribution. Our goal in this paper is to do a relatively complete investigation on this topic. For this purpose, we consider the extensions of both LTL and CTL with variable quantifications (denoted by VLTL and VCTL) where the variable quantifiers can be nested arbitrarily with temporal and Boolean operators, and the quantifications are not over the set of data values occurring before or after the current position, but over the full data domain. In addition, we investigate the decidability and complexity of both the satisfiability and the model checking problem. More specifically, we obtain the following results.

1. Existential variable quantifiers or one single universal quantifier in the beginning already entails undecidability for the satisfiability problem of both VLTL and VCTL.
2. If the existential variable quantifiers are not nested, then the satisfiability problem becomes decidable for both VLTL and VCTL. The proof is obtained by a reduction to the nonemptiness problem of alternating one register automata ([8]).

Table 1 Summary of the results: U: Undecidable, D: Decidable, T: Theorem, C: Corollary.

	$\exists^*$-VLTL	$\forall^*$-VLTL	NN-$\exists^*$-VLTL	NN-$\forall^*$-VLTL	$\exists^*$-VLTL$_{pnf}$
SAT	U (T. 3)	U (T. 6)	D (T. 15)	U (T. 6)	D (T. 22, [10])
MC	U (C. 7)	U (C. 4)	U (C. 7)	D (C. 16)	U (C. 7)
	$\forall^*$-VLTL$_{pnf}$	$\exists$-VLTL$_{pnf}$	$\forall$-VLTL$_{pnf}$	$\exists$-VLTL$^{gdap}_{pnf}$	$\forall$-VLTL$^{gdap}_{pnf}$
SAT	U (T. 6)	D (T. 22, [10])	U (T. 6)	D (T. 22, [10])	D (T. 26)
MC	D (T. 22, [10])	U (C. 7)	D (T. 22, [10])	D (C. 27)	D (T. 22, [10])
	$\exists\forall$-VLTL$^{noap}_{pnf}$	$\forall\exists$-VLTL$^{noap}_{pnf}$	$\exists\exists$-VLTL$^{noap}_{pnf}$	$\forall\forall$–VLTL$^{noap}_{pnf}$	
SAT	U (T. 10)	U (T. 10)	D (T. 22, [10])	?	
MC	U (C. 12)	U (C. 12)	U (T. 9)	D (T. 22, [10])	
	$\exists^*$-VCTL	$\forall^*$-VCTL	NN-$\exists^*$-VCTL	NN-$\forall^*$-VCTL	$\exists^*$-VLTL$_{pnf}$
SAT	U (C. 5)	U (C. 8)	D (T. 17)	U (Cor. 8)	D (T. 23)
MC	U (T. 9)	U (T. 9)	U (T. 9)	D (Cor. 19)	U (T. 9)
	$\forall^*$-VCTL$_{pnf}$	$\exists$-VCTL$_{pnf}$	$\forall$-VCTL$_{pnf}$	$\exists$-VCTL$^{gdap}_{pnf}$	$\forall$-VCTL$^{gdap}_{pnf}$
SAT	U (C. 8)	D (T. 23)	U (C. 8)	D (T. 23)	?
MC	D (T. 24)	?	D (T. 24)	?	D (T. 24)
	$\exists\forall$-VCTL$^{noap}_{pnf}$	$\forall\exists$-VCTL$^{noap}_{pnf}$	$\exists\exists$-VCTL$^{noap}_{pnf}$	$\forall\forall$-VCTL$^{noap}_{pnf}$	EVCTL
SAT	U (C. 11)	U (C. 11)	D (T. 23)	?	D (T. 20)
MC	U (T. 9)	U (T. 9)	U (T. 9)	D (T. 24)	U (T. 9)

3. If only existential path quantifiers are used in VCTL, then the satisfiability problem is decidable (NEXPTIME), no matter whatever variable quantifiers are available. This result is proved by a small model property.
4. For the fragment of VLTL with one single universal variable quantifier in the beginning, if the occurrences of the non-parameterized atomic propositions are guarded by the positive occurrences of the universally quantified variable, then the satisfiability problem becomes decidable. The proof is obtained by a reduction to the nonemptiness of extended data automata ([1]). This decidability result is tight in the sense that adding one more existential variable quantifier before or after the universal one implies undecidability.
5. Based on the above results of the satisfiability problem, we also deduce the (un)decidability results of the model checking problem.

The results obtained in this paper are summarized into Table 1 (where $\exists, \forall$ mean existential and universal variable quantifier, pnf means prenex normal form, NN means non-nested, the question mark means that the decidability is open). The reader can refer to Section 2 for the definitions of the fragments of VLTL and VCTL.

Related Work. Emerson and Namjoshi proposed indexed CTL$^*\backslash X$ to specify and reason about parameterized systems ([7]). The indexed CTL$^*\backslash X$ formulas they considered are in prenex normal form and their main goal is to prove the "cutoff" results. Song and Touili considered some extensions of LTL and CTL to detect the malware where the variable quantifications can be nested arbitrarily with the other operators, but the variables range over a finite domain ([20, 19]). The temporal logics extended with variable quantifications ranging over an infinite data domain are the formalisms over infinite alphabets. Researchers have proposed many such formalisms. To cite a few, nondeterministic register automata ([15]), first-order logic with two variables ([2]), XPath with data value comparisons ([9]).

Very recently, independently of this work, Sheinvald et al. considered the characterization of simulation pre-order in VCTL* ([18]). VCTL* extends CTL* by unrestricted data variable quantifications, thus subsumes VLTL and VCTL considered in this paper. But they have not investigated the satisfiability and model checking problem for VCTL* yet.

The rest of this paper is organized as follows: Preliminaries are given in the next section, Section 3 and 4 present respectively the undecidability and decidability results. All the missing proofs will appear in the journal version of this paper.

2 Preliminaries

Let D be an infinite set of data values, AP a finite set of (non-parameterized) atomic propositions, and T with $AP \cap T = \emptyset$ a finite set of parameterized atomic propositions, where each of them carries one parameter (data value). Let Var be a countable set of data variables which range over D. Let $[k]$ denote the set $\{0, \dots, k-1\}$, for all $k \in \mathbb{N}$.

A *word* (resp. *data word*) w over AP (resp. $AP \cup T$) is a finite sequence from $(2^{AP})^*$ (resp. $(2^{AP \cup T \times D})^*$). Given $k \geq 1$, a k-ary *tree* is a set $Z \subseteq [k]^*$ s. t. for all $zi \in Z$: $z \in Z$ and $zj \in Z$ for all $j \in [i]$. The node ϵ is called the *root* of the tree. For every $z \in Z$, the nodes $zi \in Z$ for $i \in [k]$ are called the *successors* of z, denoted by $suc(z)$. Let $Leaves(Z)$ denote the set of leaves of Z. A *path* π of a tree Z is a set $\pi \subseteq Z$ s. t. $\epsilon \in \pi$ and $\forall z \in \pi$, either z is a leaf, or there is a unique $i \in [k]$ s. t. $zi \in \pi$. A k-ary *labeled tree* (resp. *data tree*) t over AP (resp. $AP \cup T$) is a tuple (Z, L), where Z is a k-ary tree and $L : Z \to 2^{AP}$ (resp. $L : Z \to 2^{AP \cup T \times D}$) is the labeling function. Given a labeled or data tree $t = (Z, L)$, let $z \in Z$ and π be a path of t, then t_z denotes the labeled or data subtree t rooted at z, and w_π denotes the word or data word on the path π of t. For $z \in Z$ in a tree or data tree $t = (Z, L)$, define the *tree type* of z in t, denoted by $type_t(z)$, as the set $\{l_0, \dots, l_{k-1}\}$ s. t. for every $j \in [k]$, if $zj \in Z$, then $l_j = \triangledown_j$, otherwise $l_j = \overline{\triangledown_j}$($\triangledown_j$ means the j-th child exists).

A *variable Kripke structure*[1] (VKS) $\mathcal{K}$ is a tuple $(AP \cup T, X, S, R, S_0, I, L, L')$, where AP and T are defined as above, X and S are finite sets of variables and states respectively, $R \subseteq S \times S$ is the set of edges, $S_0 \subseteq S$ is the set of initial states, I is the invariant function that assigns to each state a formula which is a positive Boolean combination of $x_i = x_j$ and $x_i \neq x_j$ for $x_i, x_j \in X$, $L : S \to 2^{AP \cup T \times X}$ is the state labeling function, $L' : R \to 2^{\{reset\} \times X}$ is the edge labeling function. Intuitively, if $(reset, x) \in L'((s, s'))$, then the value of the variable x is reset (to any value) when going from s to s'.

The set of (finite) computation traces and computation trees can be defined similar to Kripke structures. The difference is that the data values are added to positions or nodes of computation traces or trees, while respecting the state invariants and the edge labelings. Let $\mathcal{L}(\mathcal{K})$ and $\mathcal{T}(\mathcal{K})$ denote the set of computation traces and computation trees of $\mathcal{K}$ respectively.

For any VKS $\mathcal{K}$ with the set of variables X, it holds that in every computation trace w or computation tree t of $\mathcal{K}$, the number of data values occurring in each position of w or each node of t is at most $|X|$. Since we are interested in reasoning about the computations of variable Kripke structures, we restrict our attention in this paper to the language of data words (or data trees) s. t. there is a bound K satisfying that each position or node of data words or data trees carries at most K data values.

For the convenience of discussions, we represent data words with at most K data values in each position as K-attributed data words from $\left(2^{AP} \times (2^T \times D)^K\right)^*$ (If the number of data

[1] Variable Kripke structure defined here is the same as that in [10], except that the global invariants are replaced by local state invariants.

values at some position is less than K, then we just copy them). For a K-attributed data word $w = (A_0, ((B_{0,0}, d_{0,0}), \dots, (B_{0,K-1}, d_{0,K-1}))) \dots (A_n, ((B_{n,0}, d_{n,0}), \dots, (B_{n,K-1}, d_{n,K-1})))$, let $prj(w)$ denote the sequence $(A_0, (B_{0,0}, \dots, B_{0,K-1})) \dots (A_n, (B_{n,0}, \dots, B_{n,K-1}))$. Similarly, we represent data trees with at most K data values in each node as K-attributed data trees. From now on, when we say data words or data trees, we always mean K-attributed data words or data trees. Let $\eta = (A, ((B_1, d_1), \dots, (B_K, d_K))) \in 2^{AP} \times (2^T \times D)^K$, $p \in AP$ and $(\tau, d) \in T \times D$. By abuse of notations, we use $p \in \eta$ to mean $p \in A$, and use $(\tau, d) \in \eta$ to denote $(\tau, d) \in \cup_{1 \leq i \leq K} B_i \times \{d_i\}$.

The syntax of *Variable-LTL* (VLTL)[2] is defined by the following rules,

$$\varphi := p \mid \neg p \mid \tau(x) \mid \neg\tau(x) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \overline{X}\varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid \exists x \varphi \mid \forall x \varphi,$$

where $p \in AP, \tau \in T, x \in Var$, and $\overline{X}$ is the dual operator of X such that $\overline{X}\varphi \equiv \neg X \neg \varphi$.

Let $var(\varphi)$ and $free(\varphi)$ denote respectively the set of variables occurring in φ and the set of free variables of φ. VLTL formulas without free variable are called *sentences*.

VLTL formulas are interpreted over K-attributed data words. Let $w = w_0 \dots w_n \in \left(2^{AP} \times (2^T \times D)^K\right)^*$, φ a VLTL formula, $\lambda : free(\varphi) \to D$, and for every $i : 0 \leq i \leq n$, $w^i = w_i \dots w_n$. We define the satisfaction relation as $w \models_\lambda \varphi$ in an obvious way. In particular, $w \models_\lambda \forall x \varphi_1$ if for all $d \in D$, $w \models_{\lambda[d/x]} \varphi_1$, where $\lambda[d/x]$ is the same as λ, except assigning d to x; $w \models_\lambda \exists x \varphi_1$ if there is $d \in D$ s. t. $w \models_{\lambda[d/x]} \varphi_1$. If φ is a VLTL sentence, we will drop λ from $\models_\lambda$. Let $\mathcal{L}_K(\varphi)$ denote the set of K-attributed data words w satisfying φ.

Let $\mathcal{K}$ be a VKS and φ a VLTL sentence. Then $\mathcal{K}$ satisfies φ, denoted by $\mathcal{K} \models \varphi$, if for every computation trace w of $\mathcal{K}$, $w \models \varphi$.

Let $\exists^*$-VLTL (resp. $\forall^*$-VLTL) denote the set of VLTL formulas without using $\forall$ (resp. $\exists$) quantifier, NN-$\exists^*$-VLTL denote the set of $\exists^*$-VLTL formulas where no $\exists$ quantifiers are nested (in a strict sense), more precisely, for any pair of subformulas $\exists x \psi_1$ and $\exists y \psi_2$ s. t. $x \neq y$, if $\exists y \psi_2$ is a subformula of ψ_1, then $x \notin free(\psi_2)$. NN-$\forall^*$-VLTL is defined similarly. Let VLTL$_{pnf}$ denote the set of VLTL formulas in prenex normal form $\theta_1 x_1 \dots \theta_k x_k\ \psi$, where $\theta_1, \dots, \theta_k \in \{\exists, \forall\}$, and ψ is a quantifier-free VLTL formula. Suppose $\theta = \theta_1 \dots \theta_k \in \{\forall, \exists\}^*$, let θ-VLTL$_{pnf}$ denote the set of VLTL$_{pnf}$ formulas of the form $\theta_1 x_1 \dots \theta_k x_k \psi$. Note that in general VLTL formulas cannot be turned into equivalent prenex normal forms[3]. Suppose $\Theta \subseteq \{\forall, \exists\}^*$, let Θ-VLTL$_{pnf} = \cup_{\theta \in \Theta}$ θ-VLTL$_{pnf}$. For $\theta \in \{\exists, \forall\}^*$ (resp. $\Theta \subseteq \{\exists, \forall\}^*$), let θ-VLTL$^{noap}_{pnf}$ (resp. Θ-VLTL$^{noap}_{pnf}$) denote the set of θ-VLTL$_{pnf}$ (resp. Θ-VLTL$_{pnf}$) formulas without using atomic propositions from AP. Let $\theta \in \{\exists, \forall\}$. Then θ-VLTL$^{gdap}_{pnf}$ denote the set of θ-VLTL$_{pnf}$ formulas where each occurrence of the atomic propositions from AP is *guarded* by the positive occurrences of the (unique) quantified variable. More specifically, θ-VLTL$^{gdap}_{pnf}$ is the set of θ-VLTL$_{pnf}$ formulas $\theta x \psi$ s. t. for every $p \in AP$, each occurrence of p (resp. $\neg p$) in ψ is in the formula $p \wedge \tau(x)$ (resp. $\neg p \wedge \tau(x)$) for some $\tau \in T$. For example, $\forall x (open(x) \to close(x))$ is a $\forall$-VLTL$^{gdap}_{pnf}$ formula, while $\forall x (open(x) \to (p \wedge \neg write(x))\ U\ close(x))$ is not.

The syntax of variable-CTL (VCTL) is defined similarly to VLTL, by adding the path quantifiers A and E before every temporal operator in the syntax rules of VLTL.

VCTL formulas are interpreted over K-attributed data trees. The semantics of VCTL are defined similarly to VLTL. The syntactic fragments of VCTL can also be defined similarly to

[2] VLTL defined in [10] is a fragment of VLTL defined here. In addition, we disallow explicit data variable comparisons (e.g. equality and inequality). We believe that VLTL without any data variable comparison is a kind of first-order extensions of temporal logics of the minimum first-order feature.

[3] For instance, we conjecture that there are no VLTL$_{pnf}$ formulas equivalent to the VLTL formula $G(\exists x \tau(x))$. But at present we do not know how to prove this.

VLTL, with the additional distinction between EVCTL and AVCTL, that is, the fragment of VCTL using only E and A respectively. Let $\mathcal{L}_K(\varphi)$ denote the set of K-attributed data trees t satisfying φ.

Let $\mathcal{K}$ be a VKS and φ be a VCTL sentence. Then $\mathcal{K}$ satisfies φ, denoted by $\mathcal{K} \models \varphi$, if for every computation tree t of $\mathcal{K}$, $t \models \varphi$.

For a VLTL or VCTL formula φ, let $\overline{\varphi}$ denote the negation of φ, where $\overline{p} = \neg p$, $\overline{\neg p} = p$, $\overline{\tau(x)} = \neg\tau(x)$, $\overline{\neg\tau(x)} = \tau(x)$, $\overline{X\varphi_1} = X\overline{\varphi_1}$, $\overline{A\varphi_1} = E\overline{\varphi_1}$, and so on. Let $|\varphi|$ denote the size of φ, that is, the number of symbols in φ. We will use $\varphi_1 \to \varphi_2$ to mean $\overline{\varphi_1} \vee \varphi_2$. A VLTL or VCTL formula that does not contain subformulas of the form $\psi_1 \to \psi_2$ is called normalized.

We consider the following two decision problems for VLTL and VCTL.

- Satisfiability problem: Given a VLTL (resp. VCTL) sentence φ, decide whether φ is satisfiable, that is, whether there is a data word w (resp. there are $k \geq 1$ and a k-ary data tree t) s. t. $w \models \varphi$ (resp. $t \models \varphi$).
- Model checking problem: Given a VKS $\mathcal{K}$ and a VLTL/VCTL sentence φ, decide whether $\mathcal{K} \models \varphi$.

▶ Remark. We interpret VLTL and VCTL formulas over finite data words and finite data trees, and leave the investigations on infinite data words and infinite data trees as future work. The considerations of temporal logics interpreted over finite words and trees are normally motivated by the verification of safety properties of concurrent systems (cf. e.g. [17]) as well as the verification of properties of sequential programs.

We next define alternating register automata over k-ary 1-attributed data trees by adapting the definition of alternating register automata over unranked data trees in [8].

An *alternating register automaton* over k-ary 1-attributed data trees (ATRA) is a tuple $\mathcal{A} = (AP \cup T, Q, q_0, \delta)$, where AP (resp. T) is a finite set of atomic propositions (resp. parameterized atomic propositions), Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \to \Phi$ is the transition function, where Φ is defined by the following grammar[4],

$$p \mid \neg p \mid \tau \mid \neg\tau \mid \triangledown_i? \mid \overline{\triangledown_i}? \mid eq \mid \overline{eq} \mid q \vee q' \mid q \wedge q' \mid store(q) \mid guess(q) \mid \triangledown_i q,$$

where $p \in AP$, $\tau \in T$, $q, q' \in Q$, and $i \in [k]$. Intuitively, $p, \neg p, \tau, \neg\tau$ are used to detect the occurrences of (parameterized) atomic propositions. $\triangledown_i?, \overline{\triangledown_i}?$ are used to describe the types of nodes in trees, $eq, \overline{eq}$ are used to check whether the data value in the register is equal to the current one, $q \vee q'$ makes a nondeterministic choice, $q \wedge q'$ creates two threads with the state q and q' respectively, $store(q)$ stores the current data value to the register and transfers to the state q, $guess(q)$ guesses a data value for the register and transfers to the state q, $\triangledown_i q$ moves to the i-th child of the current node and transfers to the state q.

An ATRA $\mathcal{A} = (AP \cup T, Q, q_0, \delta)$ is called *alternating register automaton* over 1-attributed data words (AWRA) if $k = 1$.

The semantics of ATRA over k-ary 1-attributed data trees are defined in a completely analogous way as those of ATRA over unranked trees in [8]. Let $\mathcal{L}(\mathcal{A})$ denote the set of all k-ary 1-attributed data trees accepted by $\mathcal{A}$.

The closure properties of ATRA and the decidability of the nonemptiness of ATRA can be proved in the same way as the alternating register automata over unranked trees, by utilizing well-structured transition systems (cf. [8]).

▶ **Theorem 1.** *ATRAs are closed under intersection and union. The emptiness problem of ATRAs is decidable and non-primitive recursive.*

[4] The *spread* mechanism in [8] is dropped, since it is not used in this paper.

We also introduce extended data automata ([1]), another automata model over (1-attributed) data words. Extended data automata is an extension of the seminal model of data automata ([2]) over 1-attributed data words.

An *extended data automaton* (EDA) $\mathcal{D}$ is a tuple $(AP \cup T, \mathcal{A}, \mathcal{B})$ s. t. AP and T are as above, $\mathcal{A}$ is a nondeterministic letter-to-letter transducer from the alphabet $2^{AP} \times 2^T$ to some output alphabet Σ, and $\mathcal{B}$ is a finite automaton over $\Sigma \cup \{0\}$ (where $0 \notin \Sigma$). Let $w = (A_0, (B_0, d_0)) \dots (A_n, (B_n, d_n))$ be a 1-attributed data word. Then w is accepted by $\mathcal{D}$ if over $prj(w)$, $\mathcal{A}$ outputs a word $w' = \sigma_0 \dots \sigma_n$ over the alphabet Σ, s. t. for every data value $d \in D$, $cstr_d(w'')$ is accepted by $\mathcal{B}$, where $w'' = (\sigma_0, d_0) \dots (\sigma_n, d_n)$ and $cstr_d(w'')$ is defined as $\sigma'_0 \dots \sigma'_n$, satisfying that for every $i : 0 \leq i \leq n$, $\sigma'_i = \sigma_i$ if $d_i = d$, and $\sigma'_i = 0$ otherwise. Note that for every data value d not occurring in w, $cstr_d(w'') = 0^{n+1}$.

▶ **Theorem 2** ([2, 1])**.** *EDAs are closed under intersection and union. The nonemptiness problem of EDAs is decidable.*

3 Undecidability

This section is devoted to the various undecidability results for the satisfiability and model checking problem of VLTL and VCTL.

▶ **Theorem 3.** *The satisfiability problem of $\exists^*$-VLTL is undecidable.*

Proof sketch. We show this by a reduction from the PCP problem. We illustrate the proof by considering the special case $K = 1$. Let $(u_i, v_i)_{1 \leq i \leq n}$ be an instance of the PCP problem over an alphabet Σ. A solution of the PCP problem is a sequence of indices $i_1 \dots i_m$ s. t. $u_{i_1} \dots u_{i_m} = v_{i_1} \dots v_{i_m}$.

Let $\Sigma' = \Sigma \cup \{\overline{a} \mid a \in \Sigma\} \cup \{1, \dots, n\} \cup \{\overline{1}, \dots, \overline{n}\} \cup \{\#\}$. Then Σ' can be encoded by $\lceil \log(|\Sigma'|) \rceil$ bits. Let AP be a set of atomic propositions of size $\lceil \log(|\Sigma'|) \rceil$. For each $\sigma \in \Sigma'$, let $atom(\sigma)$ denote the subset of AP corresponding to the binary encoding of σ, and $type(\sigma)$ denote the conjunction of atomic propositions or negated atomic propositions from AP corresponding to the binary encoding of σ. For instance, if the binary encoding of σ is 10, let $AP = \{p_1, p_2\}$, then $atom(\sigma) = \{p_1\}$ and $type(\sigma) = p_1 \wedge \neg p_2$. The definition of $atom(\sigma)$ can be naturally extended to $atom(u)$ for words $u \in (\Sigma')^+$. In addition, let $T = \{\tau\}$. We use $atom'(\sigma)$ to denote $(atom(\sigma), \{\tau\})$. Similarly, the definition $atom'(\cdot)$ can be extended to words $u \in (\Sigma')^+$.

We intend to encode a solution of the PCP problem, say $i_1 \dots i_m$, as the 1-attributed data word over $2^{AP} \times 2^T$ of the form $w_{i_1} w_{i_2} \dots w_{i_m} (atom'(\#), d)\ \overline{w_{i_1}} \dots \overline{w_{i_m}}$ s. t.

- $prj(w_{i_j}) = atom'(i_j) atom'(u_{i_j})$ and $prj(\overline{w_{i_j}}) = atom'(\overline{i_j}) atom'(\overline{u_{i_j}})$ for every $1 \leq j \leq m$,
- the two sequences of data values corresponding to respectively $atom'(i_1) \dots atom'(i_m)$ and $atom'(\overline{i_1}) \dots atom'(\overline{i_m})$ are the same,
- the two sequences of data values corresponding to respectively $atom'(u_{i_1}) \dots atom'(u_{i_m})$ and $atom'(\overline{u_{i_1}}) \dots atom'(\overline{u_{i_m}})$ are the same.

The data word encodings of the solutions of the PCP problem can be expressed by a $\exists^*$-VLTL formula φ. For instance, during the reduction, we express as the following $\exists^*$-VLTL formula φ_1 the fact that for every two consecutive occurrences of letters from $\{atom'(\sigma) \mid \sigma \in \Sigma\}$, there are two consecutive occurrences of letters from $\{atom'(\overline{\sigma}) \mid \sigma \in \Sigma\}$ with the same letters (by viewing $atom'(\sigma)$ the same as $atom'(\overline{\sigma})$) and the same data values, $\varphi_1 = G \wedge_{\sigma_1, \sigma_2 \in \Sigma} [\psi_0 \rightarrow \exists x \exists y (\psi_1 \wedge X \psi_2 \wedge F(\psi_3 \wedge X \psi_4))]$, where $\psi_0 = type(\sigma_1) \wedge X[type(\sigma_2) \vee \vee_{1 \leq j \leq n}(type(j) \wedge X type(\sigma_2)))]$, $\psi_1 = type(\sigma_1) \wedge \tau(x)$, $\psi_2 = (type(\sigma_2) \wedge \tau(y)) \vee$

$\vee_{1\leq j\leq n}[type(j) \wedge X(type(\sigma_2) \wedge \tau(y))]$, $\psi_3 = type(\overline{\sigma_1}) \wedge \tau(x)$, $\psi_4 = (type(\overline{\sigma_2}) \wedge \tau(y)) \vee \vee_{1\leq j\leq n}[type(\bar{j}) \wedge X(type(\overline{\sigma_2}) \wedge \tau(y))]$. ◀

Since a VKS can be defined to accept the set of all 1-attributed data words, we deduce the following result.

▶ **Corollary 4.** *The model checking problem of* $\forall^*$*-VLTL is undecidable.*

By adding a universal path quantifier A before every temporal operator of φ in the proof of Theorem 3, we get a reduction to the satisfiability problem of $\exists^*$-AVCTL.

▶ **Corollary 5.** *The satisfiability problem of* $\exists^*$*-AVCTL formulas is undecidable.*

By a similar reduction from the nonemptiness of two-counter machines as that in Theorem 4.1 of [8], we can show the following result.

▶ **Theorem 6.** *The satisfiability problem of* $\forall$*-VLTL*$_{pnf}$ *is undecidable.*

▶ Remark. The above theorem demonstrates that the claim in [11] that the satisfiability problem of $\exists^*\forall$-VLTL$_{pnf}$ is decidable is incorrect. We confirmed this observation in an e-mail with Grumberg, Kupferman and Sheinvald ([12]). On the other hand, we will show that the satisfiability problem of $\forall$-VLTL$^{gdap}_{pnf}$, that is, $\forall$-VLTL$_{pnf}$ formulas where all the occurrences of (negated) atomic propositions are guarded, is decidable (cf. Theorem 26).

▶ **Corollary 7.** *The model checking problem of* $\exists$*-VLTL*$_{pnf}$ *is undecidable.*

▶ **Corollary 8.** *The satisfiability problem of* $\forall$*-AVCTL*$_{pnf}$ *is undecidable.*

▶ Remark. From the undecidability result of the satisfiability of a fragment of VCTL, we do not immediately deduce the undecidability of the model checking problem of the dual fragment. The reason is that there does not exist a VKS to define the set of all K-attributed computation trees, or define the set of k-ary K-attributed data trees even for a fixed k.

▶ **Theorem 9.** *The model checking problem is undecidable for the following fragments:* $\forall^*$*-AVCTL,* $\forall^*$*-EVCTL,* $\exists\exists$*-VCTL*$^{noap}_{pnf}$*,* $\exists\forall$*-VCTL*$^{noap}_{pnf}$*,* $\forall\exists$*-VCTL*$^{noap}_{pnf}$*, NN-*$\exists^*$*-VCTL.*

Proof sketch. We prove the theorem by reductions from the satisfiability problem of $\exists^*$-VLTL and $\forall$-VLTL over 1-attributed data words.

We illustrate the proof by considering the model checking problem of $\forall^*$-AVCTL and $\exists\exists$-VCTL$^{noap}_{pnf}$. The arguments for the other cases use the similar idea.

We first consider the model checking problem of $\forall^*$-AVCTL.

Let φ be a $\exists^*$-VLTL formula over $AP \cup T$. We will construct a VKS $\mathcal{K}$ and a $\exists^*$-EVCTL formula φ' s. t. φ is satisfiable iff $\mathcal{K} \not\models \overline{\varphi'}$. Note that $\overline{\varphi'}$ is a $\forall^*$-AVCTL formula.

The idea of the reduction is as follows: We construct a VKS $\mathcal{K}$ which is a single loop (without branchings). Thus each computation tree of $\mathcal{K}$ is in fact a data word. Then we obtain from φ by adding existential path quantifiers E before every temporal operator occurring in φ (plus some other modifications) to obtain φ'. Since $\mathcal{K}$ is a structure without branchings, the satisfaction of φ' over the computation trees of $\mathcal{K}$ mimics the satisfaction of φ over data words.

Suppose $AP = \{p_1, \ldots, p_m\}$, $T = \{\tau_1, \ldots, \tau_{n-m}\}$ (where $m \leq n$), and $\tau'_0, \tau'_1 \notin AP \cup T$. Define the VKS $\mathcal{K} = (AP' \cup T', \{x\}, S, R, S_0, I, L, L')$ as follows: $AP' = \emptyset$, $T' = \{\tau'_0, \tau'_1\}$, $S = \{s_0, s_1, \ldots, s_{2n+1}\}$, $R = \{(s_i, s_{i+1 \bmod 2n+2}) \mid 0 \leq i \leq 2n+1\}$, $S_0 = \{s_0\}$, for every $s_i \in S$, $I(s_i) = true$, $L(s_0) = \{(\tau'_0, x)\}$, and $L(s_i) = \{(\tau'_1, x)\}$ for every $i : 1 \leq i \leq 2n+1$, $L'((s_i, s_{i+1 \bmod 2n+2})) = \{(reset, x)\}$ for every $i : 0 \leq i \leq 2n+1$.

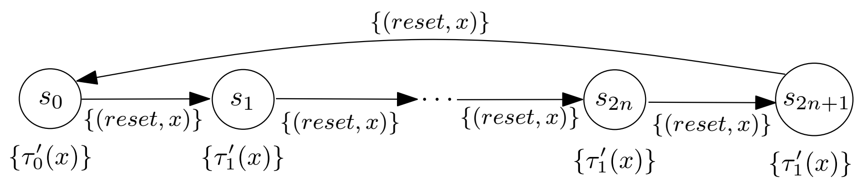


Figure 1 The Variable Kripke Structure.

Notice that in $\mathcal{K}$, T' only contains two parameterized atomic propositions. The set of atomic propositions $AP \cup T$ will be encoded by equalities and inequalities between the data values of two adjacent τ'_1-labeled positions in $\mathcal{K}$. Thus each position (A, B, d) in a 1-attributed data word over $AP \cup T$ will be encoded by a segment of computation traces in $\mathcal{K}$ of length $2n+2$ s. t. the position 0 is labeled by τ'_0, the position $(2i-1)$ and $(2i)$ encode the satisfaction on $A \cup B$ of the i-th atomic proposition from $AP \cup T$, and the last position has the data value d. In addition, x is reset on each edge $(s_i, s_{i+1 \bmod 2n+2})$ $(0 \leq i \leq 2n+1)$ so that an arbitrary data value can be assigned to x on each position $s_0, s_1, \ldots, s_{2n+1}$.

We use an example to illustrate the construction of the $\exists^*$-EVCTL formula φ' from φ. Suppose $AP = \emptyset$ and $T = \{\tau_1, \tau_2\}$ and $n = 2$. Then the $\exists^*$-EVCTL formula corresponding to the $\exists^*$-VLTL formula $\exists x G(\neg\tau_1(x) \vee XF\tau_2(x))$ is $\exists y[\tau'_0(y) \wedge \varphi'_0 \wedge \exists x EG(\psi_1 \vee (EX)^6 EF\psi_2)]$, where $\varphi'_0 = EG(\tau'_0(y) \to [(E\overline{X})^6 \tau'_0(y) \wedge EX\tau'_1(y) \wedge (EX)^3\tau'_1(y)])$ requires that the data value of the position 0 occurs in all the positions i s. t. $i \equiv 0, 1, 3 \bmod 6$, $\psi_1 = \tau'_0(y) \to [EX(\tau'_1(y) \wedge EX\neg\tau'_1(y))) \vee (EX)^5\neg\tau'_1(x)]$, and $\psi_2 = \tau'_0(y) \wedge (EX)^3(\tau'_1(y) \wedge EX\tau'_1(y)) \wedge (EX)^5\tau'_1(x)$. The formula ψ_1 is satisfied in a position if either $\tau'_0(y)$ does not occur, or $\tau'_0(y)$ occurs and one of the following conditions holds: the next two positions have different data values, or $\tau'_1(x)$ does not occur in the 5th position after the current position. Similarly for ψ_2.

For the model checking problem of $\exists\exists$-VCTL$^{noap}_{pnf}$, we reduce from the satisfiability problem of $\forall$-VLTL over 1-attributed data words. The reduction is similar to that of $\forall^*$-AVCTL. ◀

By using the similar idea as in the proof of Theorem 9, we can get the following result.

▶ **Theorem 10.** *The satisfiability problem of $\exists\forall$-VLTL$^{noap}_{pnf}$ (resp. $\forall\exists$-VLTL$^{noap}_{pnf}$) is undecidable.*

▶ **Corollary 11.** *The satisfiability problem of $\exists\forall$-VCTL$^{noap}_{pnf}$ (resp. $\forall\exists$-VCTL$^{noap}_{pnf}$) is undecidable.*

▶ **Corollary 12.** *The model checking problem of $\exists\forall$-VLTL$^{noap}_{pnf}$ (resp. $\forall\exists$-VLTL$^{noap}_{pnf}$) is undecidable.*

4 Decidability

We first present the encodings of K-attributed data words and data trees into 1-attributed ones, which will be used in the proofs of this section.

Suppose that $w = w_0 \ldots w_n$ is a K-attributed data word over $AP \cup T$ s. t. for every $i : 0 \leq i \leq n$, $w_i = (A_i, ((B_{i,0}, d_{i,0}), \ldots, (B_{i,K-1}, d_{i,K-1})))$. Let $p' \notin AP \cup T$ and $AP' = AP \cup \{p'\}$. A 1-*attributed encoding* of w, denoted by $enc(w)$, is a data word $w' = w'_{0,0} \ldots w'_{0,K-1} \ldots w'_{n,0} \ldots w'_{n,K-1}$ over $AP' \cup T$ s. t. for every $i : 0 \leq i \leq n$, $w'_{i,0} = (A_i \cup \{p'\}, (B_{i,0}, d_{i,0}))$, and for every $j : 1 \leq j \leq K-1$, $w'_{i,j} = (A_i, (B_{i,j}, d_{i,j}))$. The 1-attributed encoding of K-attributed data trees can be defined similarly. The definition of $enc(\cdot)$ can be naturally extended to the languages of K-attributed data words and data trees.

Suppose φ is a normalized VLTL formula. Then $enc(\varphi) = \varphi'_1 \wedge \varphi'_2$, where φ'_1 and φ'_2 are defined as follows.

- φ'_1 is a quantifier free VLTL formula enforcing the following constraints: p' occurs in the first position, for every occurrence of p' in some position, p' will occur in the K-th position after it if there is such a position, but does not occur in between, moreover, for every $p \in AP$, either p occurs in all the positions between two adjacent occurrences of p', or occurs in none of them.
- φ'_2 is obtained from φ by replacing X(resp. $\overline{X}$) by X^K(resp. $\overline{X}^K$), and applying some proper replacements for the (parameterized) atomic propositions. For instance, the occurrence of p_1 in Fp_1 is replaced by $p' \wedge \vee_{0 \leq i \leq K-1} X^i p_1$.

Here is an example for $enc(\varphi)$: $enc(Fp_1) = \varphi'_1 \wedge F(p' \wedge \vee_{0 \leq i \leq K-1} X^i p_1)$.

Similarly, $enc(\varphi)$ can be defined for VCTL formulas.

▶ **Proposition 13.** *For every VLTL (resp. VCTL) formula φ, $enc(\mathcal{L}_K(\varphi)) = \mathcal{L}_1(enc(\varphi))$.*

4.1 Non-nested existential variable quantifiers

▶ **Proposition 14.** *Let $\mathcal{K}$ be a VKS . Then $enc(\mathcal{L}(\mathcal{K}))$ (resp. $enc(\mathcal{T}(\mathcal{K}))$) can be defined by an AWRA (resp. ATRA).*

▶ **Theorem 15.** *The satisfiability problem of NN-$\exists^*$-VLTL is decidable and non-primitive recursive.*

Proof sketch. The decidability proof is by a reduction to the nonemptiness problem of AWRAs. Since quantifiers are not nested, w.l.o.g. we assume that there is only one variable, say x, used in φ. Note that the variable x may be reused and existentially quantified for many times. The AWRA $\mathcal{A}_{enc(\varphi)}$ can be constructed by induction on the syntax of NN-$\exists^*$-VLTL formulas similar to the construction of alternating automata from LTL formulas (cf. [21]), by using $guess(q)$ to deal with the existential quantifiers $\exists x$.

The lower bound is obtained by a reduction from the nonemptiness problem of two-counter machines with incrementing errors (cf. e.g. [6]). The reduction is similar to that in Theorem 6, with all the four conditions, except the last one, expressed in NN-$\exists^*$-VLTL. ◀

▶ **Corollary 16.** *The model checking problem of NN-$\forall^*$-VLTL is decidable and non-primitive recursive.*

▶ **Theorem 17.** *The satisfiability problem of NN-$\exists^*$-VCTL is decidable and non-primitive recursive.*

Theorem 17 is proved in the same way as Theorem 15, by utilizing the following result.

▶ **Lemma 18.** *For every $\exists^*$-VCTL sentence φ, if φ is satisfiable over a 1-attributed data tree, then there is a $(2|\varphi|)$-ary 1-attributed data tree satisfying φ.*

Similar to Corollary 16, we deduce the following result from Theorem 17.

▶ **Corollary 19.** *The model checking problem of NN-$\forall^*$-VCTL formulas is decidable and non-primitive recursive.*

4.2 Existential path quantifiers for VCTL

▶ **Theorem 20.** *The satisfiability problem of EVCTL is in NEXPTIME.*

Theorem 20 can be deduced from the following lemma.

▶ **Lemma 21.** *Let φ be an EVCTL formula, t be a data tree, and $\lambda : free(\varphi) \to D$ s. t. $t \models_\lambda \varphi$. Then a data tree t' can be constructed from (t, λ) s. t. $t' \models_\lambda \varphi$ and (t', λ) contains at most $|\varphi|$ data values.*

Proof sketch. The proof is by induction on the syntax of EVCTL formulas. The induction base $\varphi := p, \neg p, \tau(x), \neg\tau(x)$ is trivial. For induction step, we show the cases $\exists x \varphi_1$ and $\forall x \varphi_1$.

$\varphi := \exists x \varphi_1$: Suppose $t \models_\lambda \exists x \varphi_1$. Then there is $d \in D$ s. t. $t \models_{\lambda[d/x]} \varphi_1$. By the induction hypothesis, t_1 can be constructed from $(t, \lambda[d/x])$ s. t. $t_1 \models_{\lambda[d/x]} \varphi_1$ and $(t_1, \lambda[d/x])$ contains at most $|\varphi_1|$ data values. Then (t_1, λ) is the desired pair.

$\varphi := \forall x \varphi_1$: Suppose $t \models_\lambda \forall x \varphi_1$. Then for every $d \in D$, $t \models_{\lambda[d/x]} \varphi_1$. Suppose the range of λ is $\{d_1, \ldots, d_k\}$. Let d_0 be a data value not occurring in t. In addition, if $D(t) \setminus \{d_1, \ldots, d_k\}$ contains at least $|\varphi_1| - k$ data values, let $d_{k+1}, \ldots, d_{|\varphi_1|}$ be a sequence of $|\varphi_1| - k$ distinct data values from $D(t) \setminus \{d_0, d_1, \ldots, d_k\}$; otherwise let $d_{k+1}, \ldots, d_{|\varphi_1|}$ be a sequence of $|\varphi_1| - k$ distinct data values from $D \setminus \{d_0, d_1, \ldots, d_k\}$ that include all the data values in $D(t) \setminus \{d_0, \ldots, d_k\}$. Then from $t \models_{\lambda[d_i/x]} \varphi_1$ (where $i = 0, \ldots, |\varphi_1|$) and the induction hypothesis, we know that t_i can be constructed from $(t, \lambda[d_i/x])$ s. t. $t_i \models_{\lambda[d_i/x]} \varphi_1$, $(t_i, \lambda[d_i/x])$ contains at most $|\varphi_1|$ data values. Since for every $i : 0 \leq i \leq |\varphi_1|$, t_i contains at most $|\varphi_1|$ data values, we could replace the data values of t_i that are from $D \setminus \{d_0, \ldots, d_{|\varphi_1|}\}$ by data values in $\{d_1, \ldots, d_{|\varphi_1|}\}$, to get t'_i s. t. all the data values of t'_i are from $\{d_1, \ldots, d_{|\varphi_1|}\}$ and $t'_i \models_{\lambda[d_i/x]} \varphi_1$. Note that d_0 does not occur in any of $t'_0, \ldots, t'_{|\varphi|}$. Without loss of generality, we may assume that the roots of $t'_0, \ldots, t'_{|\varphi_1|}$ have the same label. Let t' be the data tree obtained from t'_0 by adding all the subtrees of the roots of $t'_1, \ldots, t'_{|\varphi_1|}$ as the new subtrees of the root of t'_0 (with the original subtrees of the root of t'_0 untouched). We claim that $t' \models_\lambda \forall x \varphi_1$. At first, for every d_i with $i : 0 \leq i \leq |\varphi_1|$, we have $t'_i \models_{\lambda[d_i/x]} \varphi_1$, thus $t' \models_{\lambda[d_i/x]} \varphi_1$ since φ_1 contains only existential path quantifiers. Let $d \notin \{d_0, \ldots, d_{|\varphi_1|}\}$. Since $t'_0 \models_{\lambda[d_0/x]} \varphi_1$ and neither d nor d_0 occurs in t'_0, assigning d to x has the same impact as assigning d_0 to x for the satisfaction of φ_1 on t'_0. Therefore, $t'_0 \models_{\lambda[d/x]} \varphi_1$. We deduce that $t \models_{\lambda[d/x]} \varphi_1$, since φ_1 contains only existential path quantifiers. From the fact that d is an arbitrary data value not in $\{d_0, \ldots, d_{|\varphi_1|}\}$, we conclude that $t' \models_\lambda \forall x \varphi_1$ and (t', λ) contains at most $|\varphi_1| + 1 \leq |\varphi|$ data values. ◀

4.3 Variable quantifications in the beginning

▶ **Theorem 22.** *([10]) The following two problems are PSPACE-complete: The satisfiability problem of $\exists^*$-VLTL$_{pnf}$ and the model checking problem of $\forall^*$-VLTL$_{pnf}$.*[5]

▶ **Theorem 23.** *The satisfiability problem of $\exists^*$-VCTL$_{pnf}$ is EXPTIME-complete.*

The proof of the upper bound is similar to the proof of the satisfiability problem of $\exists^*$-VLTL$_{pnf}$. The lower bound follows from the satisfiability problem of CTL.

▶ **Theorem 24.** *The model-checking problem of $\forall^*$-VCTL$_{pnf}$ is decidable in EXPTIME.*

[5] In [10], only model checking problem of $\forall^*$-VLTL$_{pnf}$ is considered. The result for the satisfiability problem of $\exists^*$-VLTL$_{pnf}$ can be shown by following the same idea.

Theorem 24 can be easily deduced from the following lemma.

▶ **Lemma 25.** *Let $\mathcal{K} = (AP, X, S, R, S_0, I, L, L')$ be a VKS and $\forall x_1...\forall x_n\psi$ be a $\forall^*$-$VCTL_{pnf}$ sentence. Then there is a computation tree $t = (Z, L)$ of $\mathcal{K}$ s. t. $t \models \exists x_1...\exists x_n\overline{\psi}$ iff there is a computation tree $t' = (Z, L')$ of $\mathcal{K}$ s. t. $t' \models \exists x_1...\exists x_n\overline{\psi}$ and t' contains at most $|X| + n$ different values.*

We next consider the satisfiability and model checking problem of $\forall$-VLTL$^{gdap}_{pnf}$.

▶ **Theorem 26.** *The satisfiability problem of $\forall$-$VLTL^{gdap}_{pnf}$ is decidable.*

Proof sketch. Suppose $\varphi = \forall x\psi$ is a $\forall$-VLTL$^{gdap}_{pnf}$ formula over $AP \cup T$.

From the definition of $enc(\cdot)$, we know that $enc(\varphi) = \varphi'_1 \wedge \varphi'_2$ and $\varphi'_2 = \forall x\psi'$ for some quantifier free VLTL formula ψ'. Then $enc(\varphi)$ can be rewritten into $\forall x(\varphi'_1 \wedge \psi')$, since no variables occur in φ'_1. So $enc(\varphi)$ is a $\forall$-VLTL$_{pnf}$ formula over $AP' \cup T$, where $AP' = AP \cup \{p'\}$.

It is not hard to observe that if φ is a $\forall$-VLTL$^{gdap}_{pnf}$ formula, then ψ' can be rewritten into a quantifier free VLTL formula where all the occurrences of p and $\neg p$ for $p \in AP$ are guarded by the positive occurrences of $\tau(x)$ for some $\tau \in T$. For instance, an occurrence of $p \wedge \tau(x)$ in ψ s. t. $p \in AP$ and $\tau \in T$ is transformed into $(p' \wedge \bigvee\limits_{0\leq i\leq K-1} X^i p) \wedge (p' \wedge \bigvee\limits_{0\leq i\leq K-1} X^i\tau(x))$, which is equivalent to $p' \wedge \bigvee\limits_{0\leq i\leq K-1} X^i(p \wedge \tau(x))$, since either none of $X^i p$ holds or all of them hold. By abuse of notations, we still denote the resulting formula by ψ'. Note that the formula $\forall x(\varphi'_1 \wedge \psi')$ is not a $\forall$-VLTL$^{gdap}_{pnf}$ formula since the occurrences of p' are not guarded.

To continue the proof, we introduce the following notation. Suppose $w = w_0 \dots w_n$ is a 1-attributed data word over $AP' \cup T$ s. t. $w_i = (A_i, (B_i, d_i))$ for every $i : 0 \leq i \leq n$. Then $prj_{AP}(w) = w_0|_{AP} \dots w_n|_{AP}$, where for every $i : 0 \leq i \leq n$, $w_i|_{AP} = (A_i \cap AP, (B_i, d_i))$. The definition of $prj_{AP}(\cdot)$ can be naturally generalized to languages of 1-attributed data words.

From Proposition 13, we know that the satisfiability of φ over K-attributed data words is reduced to the nonemptiness of the language $\mathcal{L}_1(enc(\varphi))$. The nonemptiness of $\mathcal{L}_1(enc(\varphi))$ is then reduced to the nonemptiness of $prj_{AP}(\mathcal{L}_1(enc(\varphi)))$.

In addition, it is not hard to show that an EDA $\mathcal{D}_{enc(\varphi)}$ can be constructed from $enc(\varphi)$ s. t. $\mathcal{L}(\mathcal{D}_{enc(\varphi)}) = prj_{AP}(\mathcal{L}_1(enc(\varphi)))$. The decidability then follows from the decidability of the nonemptiness of EDA. ◀

▶ **Corollary 27.** *The model checking problem of $\exists$-$VLTL^{gdap}_{pnf}$ is decidable.*

References

1 R. Alur, P. Cerny, and S. Weinstein. Algorithmic analysis of array-accessing programs. *ACM Trans. Comput. Logic*, 13(3):27:1–27:29, 2012.

2 M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4), 2011.

3 E. Damaggio, A. Deutsch, R. Hull, and V. Vianu. Automatic verification of data-centric business processes. In *BPM*, 2011.

4 N. Decker, P. Habermehl, M. Leucker, and D. Thoma. Ordered navigation on multi-attributed data words. In *CONCUR*, 2014.

5 S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *LICS*, 2013.

6 S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10(3):16:1–16:30, 2009.

7 E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.

8 D. Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012.

9 D. Figueira. Decidability of downward XPath. *ACM Trans. Comput. Log.*, 13(4):34, 2012.

10 O. Grumberg, O. Kupferman, and S. Sheinvald. Model checking systems and specifications with parameterized atomic propositions. In *ATVA*, 2012.

11 O. Grumberg, O. Kupferman, and S. Sheinvald. An automata-theoretic approach to reasoning about parameterized systems and specifications. In *ATVA*, 2013.

12 O. Grumberg, O. Kupferman, and S. Sheinvald. Personal communication, June 2014.

13 I. M. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable and undecidable fragments of first-order branching temporal logics. In *LICS*, 2002.

14 I. M. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Logic*, 106(1-3):85–134, 2000.

15 M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

16 A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. In *FSTTCS*, 2010.

17 O. Kupferman. Variations on safety. In *TACAS*, 2014.

18 O. Sheinvald, S. Grumberg and O. Kupferman. A game-theoretic approach to simulation of data-parameterized systems. In *ATVA*, 2014.

19 F. Song and T. Touili. LTL model-checking for malware detection. In *TACAS*, 2013.

20 F. Song and T. Touili. Pushdown model checking for malware detection. *STTT*, 16(2):147–173, 2014.

21 M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Conference on Logics for Concurrency : Structure Versus Automata*, pages 238–266, 1996.

22 V. Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT*, 2009.

Generalized Data Automata and Fixpoint Logic*

Thomas Colcombet and Amaldev Manuel

LIAFA, Université Paris-Diderot
{thomas.colcombet, amal}@liafa.univ-paris-diderot.fr

Abstract

Data ω-words are ω-words where each position is additionally labelled by a data value from an infinite alphabet. They can be seen as graphs equipped with two sorts of edges: 'next position' and 'next position with the same data value'. Based on this view, an extension of Data Automata called Generalized Data Automata (GDA) is introduced. While the decidability of emptiness of GDA is open, the decidability for a subclass class called Büchi GDA is shown using Multicounter Automata. Next a natural fixpoint logic is defined on the graphs of data ω-words and it is shown that the μ-fragment as well as the alternation-free fragment is undecidable. But the fragment which is defined by limiting the number of alternations between future and past formulas is shown to be decidable, by first converting the formulas to equivalent alternating Büchi automata and then to Büchi GDA.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Data words, Data Automata, Decidability, Fixpoint Logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.267

1 Introduction

Data words are words that can use symbols ranging over an infinite alphabet of 'data values'. Data values are meant to be tested for equality only. Hence, one is typically interested in languages such as 'no data value appears twice', or 'all consecutive data values in the word are distinct', etc. We can already see in these examples one specificity of data words, which is that the exact domain of data values does not matter, and these can be permuted without affecting the membership to a language.

Data values are particularly interesting in several modelling contexts. In particular, data values can be understood as identifiers in a database. The exact content of an identifier does not really matter. What is interesting is to be able to refer easily to the other places in the database/document where this identifier occurs. Another situation in which the data abstraction particularly makes sense is when considering the log of a system, say a server [1]. Such a log is a sequence (potentially infinite) of events that are generated by the different clients. The events produced by the various clients can be interleaved in any manner. Hence, a standard language theoretic approach does not help in verifying meaningful properties of such a log. Indeed, if the events of the sequence are anonymous – in the sense that the identity of the client that has produced it is not retained – then the interleaving obfuscates all relevant behavior of a specific client. Data languages, by annotating each action in this sequence by the unique identifier (the data) representing the client that has produced this action, give access to much more precise information. An interesting way to analyze the structure of the log is then the ability to navigate in its structure. Properties that we are

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454.

interested to express would typically consists of combinations of queries such as 'what is the next event in the log?', 'what is the next event in the log generated by the same user?', 'what is the last event that the client has generated?', 'has this client ever generated a given event before?', etc.

There are many different formalisms for describing properties of data words, i.e., for defining data languages. They include Data Automata [3], Register Automata [13, 7], Pebble Automata [17], Class Memory Automata [1], Class Automata [4], Walking Automata [16], Variable Automata [11], First-Order logic with two variables [3], Monadic Second Order logic [5], DataLTL [14], Freeze-LTL [7] and Freeze-μ [12], Logic of Repeating Values [6], XPath [8, 9], Regular expressions [15], Data Monoids [2], among others. As opposed to the case of the classical theory of regular languages, none of these formalisms can be considered to be a faithful data word counterpart of the notion of regular languages. This is due to the fact that undecidability arises very quickly in this context, and that many formalisms that turn out to be equivalent for standard words happen to have distinct expressiveness in the case of data words (a typical example is — data monoids [2], deterministic register automata and non-deterministic register-automata [13], that all have different expressiveness). In this contribution we are more interested in the kind of formalisms following the temporal logic approach. Temporal logics (LTL, CTL, CTL* and the μ-calculus) are formalisms that can describe properties of graphs (Kripke structures), by using operators that 'walk' in the structure, and can use all the Boolean connectives. This approach is particularly suitable for instance when one is interested in analysing the log of a system as described above: basic walking constructs are 'go to the next event', 'go to the next event of the same client', 'go to the previous event', and 'go to the previous event of the client'. More complex properties have also to be expressible such as 'go to the first event generated by the client'. Such advanced navigation can be achieved either using dedicated constructs (such as the 'since' and 'until' modalities of LTL), or using explicit fixpoints as done in μ-calculus. In practice, a data word consists of a linear order of positions together with an equivalence relation expressing that two given positions in the word carry the same data values (i.e., a binary relation that expresses that two events were generated by the same client). The walking modalities are then 'next', 'previous' (that we call the global modalities), 'next in the same class', and 'previous in the same class' (that we call the class modalities).

Formalisms that describe properties of data languages using temporal logics have been introduced in [7] and [14]. These two incomparable formalisms, namely DataLTL and Freeze-LTL, are related to two well-studied notions of automata, respectively Data Automata [3] and Register Automata[13, 7]. The logic in this paper is a notion along the lines of DataLTL. It is subsumed by Freeze-μ (which is undecidable over data ω-words) and is incomparable with the logics in [7, 6]. DataLTL is equipped with the four modalities described above, as well as until and since operators that can be used either with respect to global modalities or class modalities. Satisfiability of this logic is decidable by reduction to the decidability of the emptiness of Data Automata. This work was itself a continuation of another one [3] in which the satisfiability of first-order with two variables is shown, and Data Automata are introduced for this purpose. Though this logic is not syntactically a temporal logic, its behavior is in fact the one of a temporal logic.

Contribution. Our contributions are two fold. First, we introduce a generalization of Data Automata, called Generalized Data Automata. While the emptiness problem of GDA is open, we prove the decidability of a subclass of automata, namely the class of Büchi GDA via a reduction to Multicounter Automata. Secondly we generalize the notion of DataLTL by introducing a natural fixpoint logic. It is shown that the μ-fragment, as well as alternation-free fragment, of this logic is undecidable. For this reason, we restrict our attention to the class of formulas in which the alternation between backward and forward modalities is bounded (this can be syntactically enforced very easily). It is shown that the satisfiability of the alternation-free fragment of this subclass is decidable by first translating the formula into an alternating automaton and then by simulating the alternating automaton by a Büchi GDA using games.

Organization of the paper. In Section 2 we introduce the basics of data ω-words and languages. In Section 3 we introduce generalized data automata and discuss their closure properties and subsequently prove the decidability of the emptiness problem for Büchi GDA. In Section 4 we define μ-calculus on data words and introduce the bounded-reversal alternation-free fragment. We then introduce alternating parity automata and prove the simulation theorem, which is followed by the decidability of the bounded reversal alternation-free fragment. Finally in Section 5 we discuss future work and conclude.

2 Data ω-words and Data Automata

We begin by recalling the basics of data words and Data Automata. Let Σ be a finite alphabet of *letters* and $\mathcal{D}$ be an infinite set. The elements of $\mathcal{D}$, often denoted by d_1, d_2, etc., are called *data values.* A *data word* is a finite sequence of pairs from the product alphabet $\Sigma \times \mathcal{D}$. Likewise a *data ω-word* is a sequence of length ω of pairs from $\Sigma \times \mathcal{D}$. A data language is a set of data words and likewise a data ω-language is a set of data ω-words.

We recall some standard notions related to data words. Let $w=(a_1,d_1)(a_2,d_2)\ldots(a_n,d_n)$ be a data word. The data values impose a natural equivalence relation $\sim$ on the positions in the data word, namely positions i and j are equivalent, i.e. $i \sim j$, if $d_i = d_j$. An equivalence class of the relation $\sim$ is called simply a *class.* The set of all positions in a data word is partitioned into classes. The *global successor* and *global predecessor* of a position i are the positions $i+1$ and $i-1$ respectively (if they exist). For convenience we use $g(i)$ and $g^{-1}(i)$ to denote the global successor and global predecessor of position i. The *class successor* of a position i (if it exists), denoted as $c(i)$, is the leftmost position after i in its class. Symmetrically *class predecessor* of a position i (if it exists), denoted as $c^{-1}(i)$, is the rightmost position before i in its class. These notions are naturally extended to the case of data ω-words.

To simplify the discussion we assume that *all classes in a data ω-word are infinite.* This assumption is similar to the one on infinite trees (that all maximal paths are infinite); by this assumption global successor and class successor relations become total functions. All the results presented later hold without this proviso as well.

Next we recall the notion of Data Automaton (DA for short) introduced in [3]. Originally it is formulated as a composition of two finite state automata. The definition here follows an equivalent presentation due to [1]. Intuitively it is a finite state machine that reads input pairs from $\Sigma \times \mathcal{D}$ and updates the state as follows. During the run the state after reading the pair at position i depends on the state at the class predecessor position of i in addition to the state and input letter at the position i. Formally a Data Automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, I, F_c, F_g)$ where Q is a finite set of states, Σ is the finite alphabet, $\Delta \subseteq Q \times (Q \cup \{\perp\}) \times \Sigma \times Q$ is the transition relation, I is the set of initial states, F_c is the set of class Büchi states, and F_g is the set of global Büchi states.

Next we define the run of a Data Automaton. A run $\rho \in (Q \times \mathcal{D})^\omega$ of the automaton

$\mathcal{A}$ on a data ω-word $w = (a_1, d_1)(a_2, d_2)\ldots$ is a relabelling of w by the states in Q, i.e. $\rho = (q_1, d_1)(q_2, d_2)\ldots$ such that the tuple $(q_0, \perp, a_1, q_1)$ is a transition in Δ for some $q_0 \in I$ and, for each position $i > 1$ with a class predecessor, say j, the tuple (q_{i-1}, q_j, a_i, q_i) is a transition in Δ, otherwise if $i > 1$ does not have a class predecessor, then the tuple $(q_{i-1}, \perp, a_i, q_i)$ is in Δ. The run ρ is *accepting* if there is a global Büchi state that occurs infinitely often in the sequence $q_1 q_2 \ldots$, and for every class $\{i_1, i_2, \ldots\}$ there is a class Büchi state occurring infinitely often in the sequence $q_{i_1} q_{i_2} \ldots$. The data ω-word w is accepted if the automaton $\mathcal{A}$ has an accepting run on w. The set of all data ω-words accepted by the automaton $\mathcal{A}$ is called the language of $\mathcal{A}$.

3 Generalized Data Automata

In this section we introduce a generalization of Data Automaton. For this purpose we view a data ω-word as a directed graph with positions as vertices and the global successor and class successor relations as edges. For convenience we refer to these edges as global and class edges. Since both global successor and class successor relations are functions any path in this graph is completely specified by the starting position and a sequence over the alphabet $\{g, c\}$ denoting which edge is taken. Formally a path $\pi = e_1 e_2 \ldots e_n \in \{g, c\}^*$ from the position i connects the sequence of vertices $i, e_1(i), e_2(e_1(i)), \ldots e_n(\ldots e_1(i))$. Similarly an infinite path is an ω-sequence over the alphabet $\{g, c\}$.

A given run of the Data Automaton is accepted or rejected based on two ω-regular conditions; one on the global path (composed only of global edges) and one on each class (composed only of class edges). Next we introduce a generalization of Data Automaton where an ω-regular condition is checked on all paths.

First we need the following definition. Let $w = (a_1, d_1)(a_2, d_2)\ldots$ be a data ω-word and $\pi = e_1 e_2 \ldots \in \{g, c\}^\omega$ be an infinite path starting from the first position. Let $i_0 = 1, i_1, i_2, i_3, \ldots$ be the sequence of positions that lie along the path π. The *path projection* of the data ω-word w w.r.t. the path π is the ω-word $a_{i_0} a_{i_1} a_{i_2} \ldots$. The *marked path projection* of the data ω-word w w.r.t. the path π, denoted as $mpp_w(\pi) \in (\Sigma \times \{\epsilon, g, c\})^\omega$, is obtained by annotating the path projection of w w.r.t. π by the path π, that is to say

$$mpp_w(\pi) = \begin{pmatrix} a_{i_0} \\ \epsilon \end{pmatrix} \begin{pmatrix} a_{i_1} \\ e_1 \end{pmatrix} \begin{pmatrix} a_{i_2} \\ e_2 \end{pmatrix} \ldots$$

Next we introduce the notion of Generalized Data Automaton that has the same transition structure as that of a Data Automaton but a more general acceptance criterion. A *generalized data automaton* $\mathcal{A}$ (for short GDA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, I, L)$ where Q is the finite set of states, Σ is the finite alphabet, $\Delta \subseteq Q \times (Q \cup \{\perp\}) \times \Sigma \times Q$ is the transition relation, and I is the set of initial states and $L \subseteq (Q \times \{\epsilon, g, c\})^\omega$ is an ω-regular language.

Given a data ω-word $w = (a_1, d_1)(a_2, d_2)\ldots$ a run $\rho \in (Q \times \mathcal{D})^\omega$ of the automaton $\mathcal{A}$ on w is a relabelling $(q_1, d_1)(q_2, d_2)\ldots$ of w with states in Q that obeys all the consistency conditions as in the case of Data Automaton. The only difference is in the criterion of acceptance. The run ρ is accepting if for all paths π in the data ω-word ρ, the marked path projection $mpp_\rho(\pi)$ is in L. The set of all data ω-words on which $\mathcal{A}$ has an accepting run is called the language of $\mathcal{A}$.

The definition of GDA presented above is not concrete, however the acceptance criterion L can be presented as a Büchi automaton which we recall next. A Büchi automaton $\mathcal{B}$ is a tuple (S, A, T, s_{in}, G) where S is a finite set of states, A is the input alphabet, $T \subseteq S \times A \times S$ is the transition relation, s_{in} is the initial state and G is the set of Büchi states. A run r

of the automaton $\mathcal{B}$ on an ω-word $a_1a_2\ldots \in A^\omega$ is a sequence of states $s_0s_1\ldots \in Q^\omega$ such that $s_0 = s_{in}$ and for each $i \in \mathbb{N}$ the tuple (s_{i-1}, a_i, s_i) is a transition in T. The run r is accepting if there is a state in G that occurs infinitely often in it. To finitely present the GDA $\mathcal{A}$ it is enough to provide a Büchi automaton over the alphabet $Q \times \{\epsilon, g, c\}$ that accepts the language L. Next we introduce an important subclass of GDA, namely the class of *Büchi* GDA. A Büchi GDA is a special case of GDA where the acceptance criterion L is an ω-regular language that is furthermore accepted by a deterministic Büchi automaton; a deterministic Büchi automaton is a Büchi automaton whose transition relation T is a function, i.e. $T : S \times A \to S$. By definition Büchi GDA are subsumed by GDA. Our next lemma says that for every Data Automaton there is an equivalent Büchi GDA (hence a GDA as well).

▶ **Lemma 1.** *For every Data Automaton there is an equivalent Büchi GDA.*

In the following we briefly discuss the closure properties of GDA and Büchi GDA. The class of data languages accepted by Data Automata are closed under union, intersection (but not under complementation). The class of languages accepted by GDA and Büchi GDA also exhibit similar closure properties. The union of two GDA (as well as Büchi GDA) accepted languages is recognized by the disjoint union of the respective (Büchi) GDA. Closure under intersection is by usual product construction. (Both GDA and Büchi GDA are not closed under complementation, this follows from the fact that over finite data words GDA are equivalent to Data Automata.)

Two additional closure properties that are relevant for GDA (as well as for DA) are the closure under renaming and closure under composition which we recall now. For a map $h : \Sigma \to \Gamma$ and a data ω-word w over $\Sigma \times \mathcal{D}$, the renaming of w under h, denoted as $h(w)$, is obtained by replacing each letter $a \in \Sigma$ occurring in w by $h(a)$. For a language L of data ω-words over $\Sigma \times \mathcal{D}$, the renaming of L under h, in notation $h(L)$, is simply the set of all renamings $h(w)$ of each word $w \in L$.

Assume A, B, C are letter alphabets. A GDA over the alphabet $(A \times B) \times \mathcal{D}$ can be thought of as a machine that reads a data ω-word over the alphabet $A \times \mathcal{D}$ and applying a labelling of each position by a letter from the set B. In other words the machine can be thought of as a letter-to-letter transducer. The composition of languages correspond to the operation of cascading (feeding the output label of one machine into the input of another) the respective automata. Let L_1 and L_2 be two data ω-languages over the alphabets $(A \times B) \times \mathcal{D}$ and $(B \times C) \times \mathcal{D}$ respectively. The composition $Comp(L_1, L_2)$ of L_1 and L_2 is the set of data ω-words $((a_1, c_1), d_1), ((a_2, c_2), d_2)\ldots$ over the alphabet $(A \times C) \times \mathcal{D}$ such that there exists a data ω-word $((a_1, b_1), d_1), ((a_2, b_2), d_2)\ldots \in (A \times B) \times \mathcal{D}$ in L_1 and $((b_1, c_1), d_1), ((b_2, c_2), d_2)\ldots \in (B \times C) \times \mathcal{D}$ in L_2. The closure of GDA and Büchi GDA under renaming and composition can be shown by standard constructions (renaming of transitions and product construction respectively) as in the case of finite state automata. The following lemma summarizes the closure properties discussed above.

▶ **Lemma 2.** *GDA as well as Büchi GDA are closed under union, intersection, renaming and composition.*

3.1 Emptiness of Büchi GDA

The rest of this section is devoted to the emptiness problem of GDA, namely *is the language of a given GDA empty?*. We don't know if the emptiness of GDA is decidable. However, by extending the decidability proof of emptiness problem of Data Automata it can be shown that the emptiness problem for Büchi GDA is decidable. As in the case of Data Automata [3], the emptiness problem of GDA is reduced to the emptiness problem of Multicounter Automata which is decidable.

The general idea is as follows. Given a Büchi GDA $\mathcal{A}$ we construct a Multicounter Automaton that guesses a data ω-word w and simulates the automaton $\mathcal{A}$ on w and accepts if and only if $\mathcal{A}$ accepts w. Since a data ω-word is an infinite object the Multicounter Automaton cannot guess the whole word w. Instead it guesses a finite data word satisfying certain conditions that guarantees the existence of a data ω-word in the language of the automaton $\mathcal{A}$.

Now we proceed with the proof. Fix a Büchi GDA $\mathcal{A} = (Q, \Sigma, \Delta, I, L)$ and a deterministic Büchi automaton $\mathcal{B} = (S, A = Q \times \{\epsilon, g, c\}, T, s_{in}, G)$ accepting the language L.

Let $w = (a_1, d_1)(a_2, d_2)\ldots$ be a data ω-word accepted by the automaton $\mathcal{A}$ and let $\rho = (q_1, d_1)(q_2, d_2)\ldots$ be a successful run of $\mathcal{A}$ on w. Therefore for every infinite path π the ω-word $mpp_\rho(\pi)$ is accepted by the Büchi automaton $\mathcal{B}$. Let π_1 and π_2 be two infinite paths. Their respective marked path projections agree on the common prefix of π_1 and π_2. Since the automaton $\mathcal{B}$ is deterministic the (unique) runs of $\mathcal{B}$ on $mpp_\rho(\pi_1)$ and $mpp_\rho(\pi_2)$ agree on the common prefix as well. This allows us to represent the runs of the automaton $\mathcal{B}$ on the marked path projections of ρ by a labelling by subsets of S in the following way.

Let $\pi = e_1 \ldots e_n \in \{g, c\}^*$ be a finite path connecting the sequence of positions $j_0 = 1$, $j_1, \ldots, j_n = i$. The marked path projection of ρ w.r.t. π is the word $(q_{j_0}, \epsilon)(q_{j_1}, e_1) \ldots (q_{j_n}, e_n)$ over the alphabet $Q \times \{\epsilon, g, c\}$. By $\mathbf{P}(S)$ we denote the power set of S. Let $S_1 S_2 \ldots \in (\mathbf{P}(S))^\omega$ be such that S_i is the set of all states q such that there is a finite path $\pi \in \{g, c\}^*$ ending in position i and the unique partial run of the automaton $\mathcal{B}$ on the marked path projection of π ends in state q. The ω-word $S_1 S_2 \ldots \in (\mathbf{P}(S))^\omega$ can be seen as the superposition runs of the automaton $\mathcal{B}$ on each of the marked string projections. We call the data word $\zeta = ((q_1, S_1), d_1)((q_2, S_2), d_2) \ldots \in ((Q \times \mathbf{P}(S)) \times \mathcal{D})^\omega$ the annotated run.

As we mentioned earlier a witness for non-emptiness of the language of the automaton $\mathcal{A}$ is an infinite object. Hence it is not possible to compute the witness algorithmically. Instead one has to define a finite object that witnesses the non-emptiness. In the case of a Büchi automaton over infinite words this finite object is a word of the form $u \cdot v$ such that $u \cdot v^\omega$ is in the language of the automaton. In the case of Büchi GDA, u and v are two finite data words such that $u \cdot v_1 \cdot v_2 \ldots$ is in the language of the automaton where $v, v_1, v_2, \ldots$ all have the same string projections and identical classes, in other words $v_1, v_2, \ldots$ are obtained from v by renaming of data values.

We fix some notation. Let $w = (a_1, d_1) \ldots (a_n, d_n)$ be a finite data word over the alphabet Σ. A position with no class successor is called a class-maximal position. Similarly a position with no class predecessor is called a class-minimal position. The class vector of w is vector $C(w) : \Sigma \to \mathbb{N}$ that maps each letter a in Σ to the number of class-maximal positions labelled by a.

Next we formally define the notion of the finite witness in the case of Büchi GDA. Let $u, v \in (\Sigma \times \mathcal{D})^*$ be two finite data words and let $w = u \cdot v$. Let $\rho = \rho_u \cdot \rho_v \in (\Delta \times \mathcal{D})^*$ be a partial run of the Büchi GDA on the finite data word w (A partial run is a finite prefix/infix/suffix of some run of the automaton under consideration). Let $\zeta = \zeta_u \cdot \zeta_v \in ((Q \times \mathbf{P}(S)) \times \mathcal{D})^*$ be the annotated run of the automaton $\mathcal{A}$ on the data word w. (Note that the definition of annotation extends to finitely data words naturally). We aim at constructing a data ω-word in the language of the automaton $\mathcal{A}$ by repeatedly appending the data word v (with possible renaming of data values) to the end of w. Therefore the 'configuration' of the automata $\mathcal{A}$ and $\mathcal{B}$, namely the states at which the partial runs of both automata end, have to be the same at the end of the data words u and w. Moreover the number of class-maximal positions in ζ_w annotated with a pair $(q, S') \in Q \times \mathbf{P}(S)$ has to be at least the number of class-maximal positions in ζ_u annotated with the same pair for the pumping to work correctly. Finally for the acceptance criterion to be satisfied every partial run of the automaton $\mathcal{B}$ on the marked path projection of ρ w.r.t a path starting from a class-maximal position in u and ending in a class-maximal position in v (including the last position) has to see a Büchi state (in G). All these conditions are summarized below;

The triple w, ρ, ζ forms a *regular witness* if the following conditions are met.

(i) The state at the end of the partial runs ρ_u and ρ_w are the same.

(ii) $S_u = S_w$ where S_u and S_w are annotations at the last positions of ζ_u and ζ_w respectively.

(iii) Let C_u and C_w be the class vectors of ζ_u and $\zeta_w$respectively. Then,

- **(a)** $C_u \leq C_w$ in the componentwise ordering,
- **(b)** for all $(q, S') \in Q \times \mathbf{P}(S)$, if $C_u((q, S')) = 0$ then it is the case that $C_w((q, S')) = 0$.
- **(c)** Every partial run of the automaton $\mathcal{B}$ on the marked path projection of ρ w.r.t a path starting from a class-maximal position in u and ending in a class-maximal position in v (including the last position) has to see a Büchi state (in G).

In the subsequent lemma we prove the necessity and sufficiency of regular witnesses for deciding the nonemptiness. The proof rests on the following two standard lemmas.

▶ **Lemma 3** (Dickson's lemma). *Fix a $k \in \mathbb{N}$. Every infinite sequence of vectors $v_0, v_1, \ldots$ where $v_i \in (\mathbb{N}_0)^k$ contains an infinite nondecreasing subsequence $v_{i_0} \leq v_{i_1} \leq \ldots$ where the ordering $\leq$ is componentwise.*

▶ **Lemma 4** (König's lemma for words). *If A is a finite set and $L \subseteq A^*$ is infinite then there exists $x \in A^\omega$ such that x has infinitely many prefixes in L.*

▶ **Lemma 5.** *Automaton $\mathcal{A}$ accepts some data ω-word if and only if there is a regular witness for the non-emptiness of $\mathcal{A}$.*

Using Lemma 5 it is possible to decide if a given GDA accepts a non-empty language. This is achieved by a reduction to the non-emptiness problem of Multicounter Automata. A *Multicounter Automata* is a finite state machine equipped with a finite set $[k]$ of counters which hold positive integer values. During each step the machine reads a letter from the input and depending on the letter just read and the current state it performs a counter action and moves to a new state. The allowed operations on the counters are *increment counter i* and *decrement counter i*, but no zero tests are allowed. During the execution if a counter holding a zero value is decremented then the machine halts erroneously. Initially the machine starts in a designated initial state with all the counters set to value zero. An execution is accepting if the machine terminates in a state which belongs to a designated set of final states with all the counters being zero. We will be crucially making use of this final zero test. Non-emptiness of Multicounter Automata is decidable which implies by virtue of the following theorem that non-emptiness of Büchi GDA is decidable.

▶ **Theorem 6.** *Given a Büchi GDA $\mathcal{A}$ one can effectively construct an exponentially-sized Multicounter Automaton which accepts a word if and only if $\mathcal{A}$ has a regular witness.*

4 μ-calculus on data ω-words

In this section, we introduce μ-calculus over data words. Let $Prop = \{p, q, \ldots\}$ be a set of propositional variables. The formulas in the logic are the following. The atomic formulas are, $p \in Prop$, $\neg p$, and $\mathcal{S}, \mathcal{P}, \mathsf{first}^c, \mathsf{first}^g$ which are zeroary modalities. Also, $\mathtt{X}^g\varphi, \mathtt{X}^c\varphi, \mathtt{Y}^g\varphi, \mathtt{Y}^c\varphi$

$$\begin{aligned}
[\![p]\!]_w &= \ell(p) & [\![\neg p]\!]_w &= \omega \setminus \ell(p)\\
[\![\mathcal{P}]\!]_w &= \{i \mid g^{-1}(i) = c^{-1}(i)\} & [\![\mathcal{S}]\!]_w &= \{i \mid g(i) = c(i)\}\\
[\![\mathsf{first}^g]\!]_w &= \{1\} & [\![\mathsf{first}^c]\!]_w &= \{i \mid \nexists j = c^{-1}(i)\}\\
[\![\mathtt{X}^g\varphi]\!]_w &= \{i \in \omega \mid g(i) \in [\![\varphi]\!]_w\} & [\![\mathtt{X}^c\varphi]\!]_w &= \{i \in \omega \mid c(i) \in [\![\varphi]\!]_w\}\\
[\![\mathtt{Y}^g\varphi]\!]_w &= \{i \in \omega \mid g^{-1}(i) \in [\![\varphi]\!]_w\} & [\![\mathtt{Y}^c\varphi]\!]_w &= \{i \in \omega \mid c^{-1}(i) \in [\![\varphi]\!]_w\}\\
[\![\mu p.\varphi]\!]_w &= \bigcap \Big\{ S \subseteq \omega \;\mid\; [\![\varphi]\!]_{w[\ell(p):=S]} \subseteq S \Big\} & [\![\varphi_1 \wedge \varphi_2]\!]_w &= [\![\varphi_1]\!]_w \cap [\![\varphi_2]\!]_w\\
[\![\nu p.\varphi]\!]_w &= \bigcup \Big\{ S \subseteq \omega \;\mid\; S \subseteq [\![\varphi]\!]_{w[\ell(p):=S]} \Big\} & [\![\varphi_1 \vee \varphi_2]\!]_w &= [\![\varphi_1]\!]_w \cup [\![\varphi_2]\!]_w
\end{aligned}$$

Figure 1 Semantics of μ-calculus on a ω-word $w = (\omega, \ell, g, c)$.

are formulas whenever φ is a formula, and $\varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2$ are formulas whenever φ_1 and φ_2 are formulas. Finally, $\mu p.\varphi, \nu p.\varphi$ are formulas whenever φ is a formula and the variable p occurs positively in φ (i.e. $\neg p$ is not a subformula of φ).

Next we disclose the semantics; as usual, on a given structure each formula denotes the set of positions where it is true. The modality first^g holds (only) on the first position and first^c holds exactly on all the first positions of classes. The modality $\mathcal{S}$ is true at a position i if the successor and class successor of i coincide. Similarly $\mathcal{P}$ is true at i if the predecessor and class predecessor of i coincide. The modalities $\mathtt{X}^g\varphi$, $\mathtt{X}^c\varphi$, $\mathtt{Y}^g\varphi$, $\mathtt{Y}^c\varphi$ hold if φ holds on the successor, class successor, predecessor and class predecessor positions respectively. Coming to the fix-point formulas, each formula $\varphi(p)$, where p occurs positively, defines a function from sets of positions to sets of positions that is furthermore monotone. We define the semantics of $\mu p.\varphi(p)$ and $\nu p.\varphi(p)$ to be the least and greatest fix-points of $\varphi(p)$ that exists due to Knaster-Tarski theorem. To formally define the semantics we consider a data ω-word as a Kripke structure $w = (\omega, \ell, g, c)$ where $\ell : \mathit{Prop} \to \mathbf{P}(\omega)$ is valuation function giving for each $p \in \mathit{Prop}$ the set of positions where p holds, g is the global successor relation and c is the class successor relation. For $S \in \mathbf{P}(\omega)$ by $w[\ell(p) := S]$ we mean w with the new valuation function ℓ' that is defined as $\ell'(p) = S$ and $\ell'(q) = \ell(q)$ for all $q \in \mathit{Prop}$, $q \neq p$. The formal semantics $[\![\varphi]\!]_w$ of a formula φ over a data word w is described in Figure 1.

Note that we allow negation only on atomic propositions. However it is possible to negate a formula in the logic. For this, define the dual modalities $\tilde{\mathtt{X}}^g$, $\tilde{\mathtt{Y}}^g$, $\tilde{\mathtt{X}}^c$, $\tilde{\mathtt{Y}}^c$ of $\mathtt{X}^g, \mathtt{Y}^g, \mathtt{X}^c, \mathtt{Y}^c$ respectively and such that $\tilde{\mathtt{M}}\varphi = \neg\mathtt{M}\neg\varphi$, where $\neg$ stands for set complement. Since successor and class successor relations are total functions it follows that $\tilde{\mathtt{X}}^g\varphi \equiv \mathtt{X}^g\varphi$, $\tilde{\mathtt{X}}^c\varphi \equiv \mathtt{X}^c\varphi$. Similarly since predecessor and class predecessor relations are partial functions it follows that $\tilde{\mathtt{Y}}^g\varphi \equiv \mathsf{first}^g \vee \mathtt{Y}^g\varphi$, $\tilde{\mathtt{Y}}^c\varphi \equiv \mathsf{first}^c \vee \mathtt{Y}^c\varphi$. To negate a formula φ we take the dual of φ; this means exchanging in the formula $\wedge$ and $\vee$, μ and ν, p and $\neg p$, and all the modalities with their dual.

If $\varphi(p_1, \ldots, p_n)$ is a formula then by $\varphi(\psi_1, \ldots, \psi_n)$ we mean the formula obtained by substituting ψ_i for each p_i in φ. As usual the bound variables of $\varphi(p_1, \ldots, p_n)$ may require a renaming to avoid the capture of the free variables of ψ_i's. For a formula φ and a position i in the word w, we denote by $w, i \models \varphi$ if $i \in [\![\varphi]\!]_w$. The notation $w \models \varphi$ abbreviates the case when $i = 1$. The *data language* of a sentence φ is the set of data words w such that $w \models \varphi$, while the *data ω-language* of a sentence φ is the set of data ω-words w such that $w \models \varphi$.

Unfortunately, even the fragment of the logic containing only μ-fixpoints itself is undecidable.

▶ **Theorem 7.** *Satisfiability of the μ-fragment is undecidable.*

This also implies the undecidability of the alternation-free fragment (recalled below). One of the sources of undecidability is the presence of both future and past modalities, or in other words the two-way-ness of the logic. Therefore we can reclaim decidability of the logic if we restrict the number of times a formula is allowed to change direction. Next we formally define this fragment, namely the *bounded reversal alternation-free fragment*. We first recall the operation of composition of formulas. Let Ψ be a set of formulas. Define the set $Comp^i(\Psi)$ inductively as $Comp^0(\Psi) = \emptyset$ and

$$Comp^i(\Psi) = \{\psi(\varphi_1, \dots, \varphi_n) \mid \psi(p_1, \dots, p_n) \in \Psi,\ \varphi_1, \dots, \varphi_n \in Comp^{i-1}(\Psi)\}\ .$$

The set of formulas $Comp(\Psi)$ is defined as $Comp(\Psi) = \bigcup_{i\in\mathbb{N}} Comp^i(\Psi)$. For a formula $\psi \in Comp(\Psi)$ we define the *Comp-height of ψ in $Comp(\Psi)$* as the least i such that $\psi \in Comp^i(\Psi)$.

For $\lambda \in \{\mu, \nu\}$ let $\mathsf{Formulas}(\lambda)$ denote the formulas which uses only the fixpoint operator λ. Then the *alternation-free* fragment, denoted as AF, is the set of formulas $\mathrm{AF} = Comp\,(\mathsf{Formulas}\,(\mu) \cup \mathsf{Formulas}\,(\nu))$; intuitively there does not exist a μ-subformula and a ν-subformula with intersecting scope in any formula of AF. We call the set of all μ-calculus formulas which does not use the modalities $\{\mathtt{Y}^c, \mathtt{Y}^g\}$ (*resp.* $\{\mathtt{X}^c, \mathtt{X}^g\}$) the forward (*resp.* backward) fragment. Forward (*resp.* backward) alternation-free fragment, denoted as $\mathrm{AF}_\mathtt{X}$ (*resp.* $\mathrm{AF}_\mathtt{Y}$) is the set of all formulas in the alternation-free fragment which are also in the forward (*resp.* backward) fragment. The *bounded reversal alternation-free fragment* of μ-calculus, denoted as BR, is the set of formulas $\mathrm{BR} = Comp(\mathrm{AF}_\mathtt{X} \cup \mathrm{AF}_\mathtt{Y})$. An example of a formula in AF but not expressible in the fragment BR (we do not prove it) is $\mu x.a \vee \mathtt{Y}^g\mathtt{X}^c x$. It tests whether an a-letter is reachable by successive steps of advancing to the next in the class, and going backward globally. An example of a formula that is in BR is $\mu y.(\nu x.a \vee \mathtt{X}^c x) \vee \mathtt{Y}^g y$.

Next we prove that the fragment BR is decidable by reducing the satisfiability problem for BR to the emptiness problem for Büchi GDA. Since both BR and Büchi GDA are closed under composition it is enough to prove that for every formula in the fragment $\mathrm{AF}_\mathtt{X}$ and $\mathrm{AF}_\mathtt{Y}$ there is a Büchi GDA that labels each position with the set of (sub)formulas true at that position.

▶ **Lemma 8.** *Given a formula φ in the backward fragment there is a Data Automaton that labels each position with the set of subformulas of φ true at that position.*

Next we show that for every formula in the forward alternation-free fragment there is a Büchi GDA that labels each position with the set of satisfied subformulas. For this, we recall the notion of alternating parity automaton over graphs (See [10] for a comprehensive presentation). First we need the basics of two player (namely *Adam* and *Eve*) games played on graphs. An arena $A = (V, E)$ is a set of positions $V = V_E \cup V_A$ partitioned into those of Adam (V_A) and those of Eve (V_E) along with a set of moves $E \subseteq (V_A \times V_E) \cup (V_E \times V_A)$ (we assume that there are no dead-ends in the game). A partial play $(v_0, v_1)(v_1, v_2) \dots (v_k, v_{k+1}) \subseteq E^*$ is a finite sequence of moves performed by the players. The position v_0 is the starting position of the play and v_{k+1} is the ending position of the play. A strategy for a player Eve (resp. Adam) σ maps a partial play ending in a position in V_E (resp. V_A) to a move in E. A play $\pi = (v_0, v_1)(v_1, v_2) \dots \in E^\omega$ is an ω-sequence of moves. We say π is a play according to the strategy σ of Eve if on all finite prefixes of π ending in V_E she plays according to σ. A winning condition $W \subseteq E^\omega$ is a set of plays which are winning for Eve. A game $\mathcal{G}$ is a triple $\mathcal{G} = (A = (V, E), v, W)$ where A is an arena, $v \in V$ is the initial position and W is the winning condition. The strategy σ is a winning strategy for Eve if all the plays according to σ are winning for Eve. The strategy is positional if for all partial plays ending on the same vertex the strategy σ agrees on the next move. A *parity game* is a game where W is presented by means of a parity condition $\Omega : V \to \{0, \dots, k\}$ for some $k \in \mathbb{N}$. Given Ω, the winning condition W is defined as the union of all plays $\pi = (v_0, v_1)(v_1, v_2)\dots$ such that the maximal number occurring infinitely often in the sequence $\Omega(v_0), \Omega(v_1), \dots$ is even. It is well-known that parity games are positionally determined. i.e. one of the players has a positional winning strategy.

Let P be a set of propositional variables. A positive conjunction $p_1 \wedge p_2 \dots \wedge p_k, k \geq 1$ over P is identified with the subset $\{p_1, \dots, p_k\}$ of P. A DNF formula over P is a disjunction $\varphi_1 \vee \varphi_2 \dots \vee \varphi_l, l \geq 1$, where each φ_j is a positive conjunction over P, which is identified with a subset of the powerset of P, namely $\{\varphi_1, \dots, \varphi_l\}$. The set of all DNF formulas over P is denoted by $\mathrm{DNF}^+(P)$. Let $\mathcal{M}$ be the set $\{\mathcal{S}, \neg\mathcal{S}, \mathcal{P}, \neg\mathcal{P}\}$. For a given a data ω-word w and a position i in w the *type* of i, denoted by $tp\,(i)$, is the subset of $\mathcal{M}$ satisfied at position i.

An *alternating parity automaton* on data ω-words $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, q_0, \Omega)$ where Q is the finite set of states, Σ is the alphabet, q_0 is the initial state, $\Delta : Q \times \Sigma \times \mathbf{P}(\mathcal{M}) \to \mathrm{DNF}^+(\{\mathtt{X}^g p, \mathtt{X}^c p \mid p \in Q\})$ is the transition relation and $\Omega : Q \to \{0, \dots, k\}$ is the parity condition. When Ω is such that all states have parity either 1 or 2 the automaton is called Büchi.

Fix an automaton $\mathcal{A}$. Given a data ω-word $w = (\omega, \lambda, g, c)$ (for convenience we let the labelling function $\lambda : \omega \to \Sigma$ map each position to its label), the acceptance of w by $\mathcal{A}$ is defined, as usual, in terms of a two-player parity game $\mathcal{G}_{\mathcal{A},w}$ (sometimes called the *membership game*) played between Adam and Eve on the arena with positions $V = V_E \cup V_A$ where $V_E = Q \times \omega$ and $V_A = \mathit{co\text{-}Dom}(\Delta) \times \omega$. The moves E are the following. On every Eve position (p, i) she can make a move to an Adam position (φ, i) where φ is a conjunction over the set $\{\mathtt{X}^g p, \mathtt{X}^c p \mid p \in Q\}$ such that $\varphi \in \Delta(p, \lambda(i), tp\,(i))$. On every Adam position (φ, i) he can make a move to an Eve position (p, j) if j is the successor (*resp.* class successor) of i and $\mathtt{X}^g p$ (*resp.* $\mathtt{X}^c p$) is in φ. Observe that there are no dead-ends in the game. The parity of the game positions are defined as follows. For all Adam positions the parity is 0 and for all Eve positions (p, i) the parity is $\Omega(p)$. We say the automaton $\mathcal{A}$ accepts the data word w if in the game $\mathcal{G}_{\mathcal{A},w}$ the player Eve has a winning strategy from the position $(q_0, 1)$.

The following lemma follows from canonical connection between μ-calculus and alternating parity automata on any fixed class of graphs ([10]).

▶ **Fact 9.** *For every formula in the forward (resp. alternation-free) fragment there is an equivalent (which is effectively obtained) alternating parity (resp. Büchi) automaton. Moreover the states of the automaton are precisely the subformulas of the given formula.*

If a data ω-word w is accepted by $\mathcal{A}$ then there is a winning strategy for Eve in the game $\mathcal{G}_{\mathcal{A},w}$ which in turn implies that Eve has a positional winning strategy for the game. A positional strategy for Eve in $\mathcal{G}_{\mathcal{A},w}$ is a function $\sigma : \omega \to (Q \to \mathit{co\text{-}Dom}(\Delta))$ such that for all i and for all $p \in Q$, $(\sigma(i))(p) \in \Delta(p, \lambda(i), tp\,(i))$. Once a strategy σ for Eve is fixed the game $\mathcal{G}_{\mathcal{A},w}$ can be seen as a game played by a single player (namely Adam) in the following way. Define $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ as the subgame where the moves of Eve are limited to $\{(p, i) \to ((\sigma(i))(p), i) \mid i \in \omega\}$. Since the moves of Eve are fixed in the game $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ $(\star)$ she wins if and only all the infinite paths in the graph $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ are winning. A *local strategy* is a partial function $f : Q \to \mathit{co\text{-}Dom}(\Delta)$ such that there exist $a \in \Sigma, \tau \in \mathbf{P}(\mathcal{M})$ such that for all $p \in \mathit{Dom}(f)$, $f(p) = \Delta(p, a, \tau)$. A local strategy f is consistent at position i if $f(p) \in \Delta(p, \lambda(i), tp\,(i))$ for all $p \in \mathit{Dom}(f)$. Observe that a positional strategy for Eve is a sequence of local strategies $(f_i)_{i \in \omega}$ such that each f_i is consistent at position i. Now we restate $(\star)$ in terms of local strategies. Let F be the set of local strategies.

A *local strategy annotation* of a data ω-word w is a sequence of local strategies $(f_i)_{i\in\omega}$ which are consistent at each position i and furthermore satisfy the following conditions. Let $(D_i)_{i\in\omega}$ be the sequence of subsets of states Q (called the set of *reachable states*) such that the local strategy f_i has domain D_i.

1. $D_1 = \{q_0\}$.
2. $q \in D_{\mathtt{M}(i)}$ iff there exists $p \in D_i$ such that $f_i(p) = \varphi$ and $\mathtt{M}q \in \varphi$ [When $\mathtt{M} = \mathtt{X}^g$ (*resp.* $\mathtt{M} = \mathtt{X}^c$) we use $\mathtt{M}(i)$ to denote the successor (*resp.* class successor) of i]. In this case we say that there is an *edge* between (p, i) and $(q, \mathtt{M}(i))$ in the strategy annotation.

A path in the strategy annotation is a sequence $(p_1, i_1) \dots (p_n, i_n)$ such that each successive tuples has an edge between them. The local strategy annotation $(f_i)_{i\in\omega}$ is *accepting* if for all infinite paths (starting from $(q_0, 1)$) it is the case that the maximal infinitely occurring parity is even.

It is straight-forward to see that Eve has a (positional) winning strategy σ in the game $\mathcal{G}_{\mathcal{A},w}$ iff all the paths in the $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ are winning iff there is a local strategy annotation in which all paths are accepting. Thus we get,

▶ **Lemma 10.** *A data ω-word w is accepted by the automaton $\mathcal{A}$ if and only if there exist a local strategy annotation $(f_i)_{i\in\omega}$ of w which is accepting.*

Next we show the goal of this section namely that for every alternating Büchi automaton there is an equivalent Büchi GDA. Since we are converting an alternating automata to a non-deterministic automata (though not of the same kind) it can be seen as an analogue of the simulation theorem for alternating tree automata. A technicality here is that in the definition of GDA we don't have access to the type of a position. Therefore the GDA has to synthesize the type of every position. This is achieved by the following lemma due to Schwentick and Björklund.

▶ **Lemma 11** ([1])**.** *There is a Data automaton $\mathcal{A}$ which reads a data ω-word and outputs the type of each position.*

Now we present the simulation theorem. The proof is using the standard technique. The GDA guesses a local strategy annotation and verifies that all paths in the annotation are accepting. The only technicality is that the automaton has to rely on the marked path projection to verify that the paths are accepting.

▶ **Proposition 12.** *Given an alternating parity (resp. Büchi) automaton $\mathcal{A}$ there is an equivalent (resp. Büchi) GDA $\mathcal{A}'$.*

Finally we prove the main theorem of this section.

▶ **Theorem 13.** *Satisfiability of bounded-reversal alternation-free μ-calculus is decidable on data ω-words.*

5 Conclusion and future work

In this paper we have introduced a generalization of Data Automata. While the emptiness problem for GDA is open it is shown that the decidability of emptiness of a subclass, namely the class of Büchi GDA, is decidable. Next, a natural fixpoint logic on data words is defined and it is shown that the μ-fragment as well as the alternation-free fragment is undecidable. Then, by limiting the number of change of directions of formulas the class of bounded reversal alternation-free fragment is defined which subsumes other logics such DataLTL and FO^2.

It is shown that satisfiability problem for the bounded-reversal alternation-free fragment is decidable by extending the results for Data automata. In fact the latter result easily extends to the case of formulas with alternation depth $\nu\mu$.

Regarding future work, there are two interesting questions; namely *the decidability of the non-emptiness problem for GDA and the satisfiability problem of the forward fragment.* However these two problems are effectively equivalent since given a GDA $\mathcal{A}$ (*resp.* Büchi) there is an effectively constructed universal parity (*resp.* Büchi) automaton $\mathcal{A}'$ accepting the accepting runs of automaton $\mathcal{A}$. It is also interesting to know if DA is strictly included in (Büchi) GDA.

References

1 H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.

2 M. Bojańczyk. Data monoids. In *STACS*, pages 105–116, 2011.

3 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.

4 M. Bojańczyk and S. Lasota. An extension of data automata that captures xpath. In *Logic in Computer Science (LICS), 2010*, pages 243–252, July 2010.

5 T. Colcombet, C. Ley, and G. Puppis. On the use of guards for logics with data. In *MFCS*, volume 6907 of *LNCS*, pages 243–255. Springer, 2011.

6 S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *Logic in Computer Science (LICS), 2013*, pages 33–42, June 2013.

7 S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), April 2009.

8 D. Figueira. Alternating register automata on finite data words and trees. *Logical Methods in Computer Science*, 8(1), 2012.

9 D. Figueira. Decidability of downward XPath. *ACM Transactions on Computational Logic*, 13(4), 2012.

10 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research.* Springer-Verlag New York, Inc., New York, NY, USA, 2002.

11 O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *Language and Automata Theory and Applications*, pages 561–572. Springer, 2010.

12 M. Jurdziński and R. Lazic. Alternating automata on data trees and xpath satisfiability. *ACM Trans. Comput. Log.*, 12(3):19, 2011.

13 M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

14 A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. In *FSTTCS*, volume 8 of *LIPIcs*, pages 481–492, 2010.

15 L. Libkin and D. Vrgoc. Regular expressions for data words. In *LPAR*, volume 7180, pages 274–288, 2012.

16 A. Manuel, A. Muscholl, and G. Puppis. Walking on data words. In *Computer Science Theory and Applications*, volume 7913 of *LNCS*, pages 64–75. 2013.

17 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. 5(3):403–435, 2004.

Consistency of Injective Tree Patterns

Claire David[1], Nadime Francis[2], and Filip Murlak[3]

1 **Université Paris-Est Marne**, Claire.David@univ-mlv.fr
2 **ENS Cachan**, francis@lsv.ens-cachan.fr
3 **University of Warsaw**, fmurlak@mimuw.edu.pl

Abstract

Testing if an incomplete description of an XML document is consistent, that is, if it describes a real document conforming to the imposed schema, amounts to deciding if a given tree pattern can be matched injectively into a tree accepted by a fixed automaton. This problem can be solved in polynomial time for patterns that use the child relation and the sibling order, but do not use the descendant relation. For general patterns the problem is in NP, but no lower bound has been known so far. We show that the problem is NP-complete already for patterns using only child and descendant relations. The source of hardness turns out to be the interplay between these relations: for patterns using only descendant we give a polynomial algorithm. We also show that the algorithm can be adapted to patterns using descendant and following-sibling, but combining descendant and next-sibling leads to intractability.

1998 ACM Subject Classification H.2.1 [Database Management]: Logical Design – Data models, F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

Keywords and phrases XML, incomplete information, injective tree patterns, consistency

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.279

1 Introduction

It is convenient to think that a database instance is a faithful representation of a fragment of reality; but, in fact, it almost never is. Pieces of information are not available, or classified, or get lost on the way due to storage and transmission failures. Additional sources of incompleteness are complex data management tasks, like data integration [11] or data exchange [6]. Since the seminal work of Imielinski and Lipski [8], incompleteness of information has been an important topic in relational database theory [7]. More recently, the need to deal with incomplete information has increased dramatically, due to large amounts of data on the Web [1]. This data tends to be more prone to errors than data stored in traditional DBMSs, and transformation, integration, and exchange of data between different applications is inherent to this context. Dealing with data on the Web also means facing new data models such as XML documents or graph databases, and scenarios involving incomplete information for such models have been considered [4, 9].

Incompleteness brings new difficulties into classical tasks such as query answering (what does it mean to answer a query over an incomplete database?), but it also gives rise to new tasks. One of such problems is consistency: is there a real instance that matches the incomplete description? A systematic study of problems related to incomplete XML data was undertaken in [2]. XML documents are modelled as unranked labelled trees. For such a tree there are several kinds of information that can be missing in the description: nodes can be missing, or their labels, or their relative position in the tree. Thus an incomplete tree can be seen as a tree with some labels missing, and some edges representing descendant relation, rather than child relation (one can also allow partial information about sibling order).

Assuming the so-called DOM semantics, nodes of XML documents have their identity, which is never lost (if it gets lost, the node is considered to be lost). On its own, such description is always consistent: we obtain a proper document by turning all edges to child edges and filling in the labels arbitrarily. Typically, however, the setting also involves a schema (DTD, XSM, RelaxNG), that describes the shape of correct documents. The structural restrictions of the schema can always be expressed by a tree automaton. Thus, the consistency problem for a fixed schema amounts to deciding if there is a tree accepted by the automaton, that matches the given incomplete description.

The incomplete descriptions of [2] coincide with the notion of tree patterns, originally introduced as an elegant formalism to express acyclic conjunctive queries over trees and extensively studied in connection with the XPath query language [3, 12, 13, 14, 15]. Our consistency problem is a variant of the satisfiability problem for tree patterns with respect to a fixed automaton [3]. The difference lies in the semantics. Classically, a pattern is satisfied in a tree if its nodes can be mapped to the tree nodes in such a way that the labels and relations are preserved. In our setting, the DOM semantics imposes an additional requirement: the mapping has to be injective. This makes the existing results on patterns inapplicable. We also cannot use the variant of the injective semantics considered in [5], where it is additionally assumed that if two pattern nodes are incomparable (neither is descendant of the other), they must be mapped to incomparable nodes in the tree.

Already in [2] it is noticed that the consistency problem is in NP, but the exact complexity is left open. For a special case of patterns (incomplete descriptions) that do not involve descendant edges, a polynomial algorithm is given. In a highly nontrivial extension of this result, Kopczynski [10] gives a polynomial procedure for patterns that contain at most one descendant edge on each branch.[1]

In this paper we close the gap: we show that the consistency problem is NP-complete. In fact our result is tight with respect to Kopczynski's polynomial algorithm: the problem is NP-hard already for patterns with at most two descendant edges per branch. We also investigate further the sources of hardness and find out that for descendant-only patterns the problem can be solved in PTIME. Finally, we consider possible extensions involving the sibling order. Combining next-sibling with descendant leads to intractability, but for patterns using only descendant and following-sibling an adaptation of our proof techniques gives tractability.

2 Preliminaries

For an unranked Σ-labelled tree T, we write $nodes_T$ for the set of nodes, $root_T$ for the root of T, and $lab_T(v)$ for the label of a node v in T. We also use the notation $u \downarrow v$ and $u \downarrow^+ v$ to indicate that node v is, respectively, a child or a descendant of node u. We write T_v for the subtree of tree T rooted at node v.

An *antichain* in a tree is any sequence of nodes such that no two of them are in the descendant relation (they can be siblings). A *frontier* is a maximal antichain that does not contain the root of the tree.

▶ **Definition 1.** A *tree pattern* π over the alphabet Σ is a finite unranked Σ-labelled tree, whose edges are of one of two kinds: child edges, denoted $\downarrow$, and descendant edges, denoted

[1] In fact, Kopczynski gives an algorithm for the general problem, but under his own semantics, resembling that of [5]. For patterns with at most one descendant per branch, this semantics coincides with the standard injective semantics.

$\downarrow^+$. We write $lab_\pi(v)$ for the label of v in π. We also use notation $u \downarrow v$ and $u \downarrow^+ v$ to indicate that the nodes are connected with a $\downarrow$-edge or $\downarrow^+$-edge, respectively.

▶ **Definition 2.** A tree pattern π is *satisfied* in a tree T, written as $T \models \pi$, if there exists an *injective homomorphism* $h\colon \pi \to T$, that is, an injective function mapping the nodes of π to nodes of T that preserves the labels and the relations, that is, for all nodes u, v in π

- $lab_T(h(v)) = lab_\pi(v)$;
- if $u \downarrow v$ in π, then $h(u) \downarrow h(v)$ in T;
- if $u \downarrow^+ v$ in π, then $h(u) \downarrow^+ h(v)$ in T.

▶ **Definition 3.** A *tree automaton* $\mathcal{A} = (\Sigma, Q, \delta, F)$ consists of an alphabet Σ, a finite set of states Q, a set of final states $F \subseteq Q$, and a transition function $\delta\colon \Sigma \times Q \to \mathcal{P}(Q^*)$, assigning regular languages over Q (represented as regular expressions) to each label and state.

A *run* of $\mathcal{A}$ over a tree T is a labelling ρ of the nodes of T with elements of Q such that for each node of v, if v has children $v_1, v_2, \dots, v_k$, then $\rho(v_1)\rho(v_2)\dots\rho(v_k) \in \delta(lab_T(v), \rho(v))$. If v is a leaf, this amounts to $\varepsilon \in \delta(lab_T(v), \rho(v))$.

A run ρ is *accepting* if the root's label is in F. If T admits an accepting run, we say that T is accepted by $\mathcal{A}$. We write $L(\mathcal{A})$ for the language recognized by $\mathcal{A}$, i.e., the set of trees accepted by $\mathcal{A}$. A state q is *productive* if it occurs in some accepting run.

Let $\mathcal{A}$ be a tree automaton. We are interested in the complexity of the following problem.

PROBLEM:	CONS$_\mathcal{A}$
INPUT:	Tree pattern π.
QUESTION:	Is there a tree $T \in L(\mathcal{A})$ such that $T \models \pi$?

Note that the automaton $\mathcal{A}$ is not part of the input. The complexity is measured in terms of the size of the pattern π. In the context of the incomplete information scenario, where π represents information about an XML document, this corresponds to *data complexity* of consistency.

3 NP-hardness

We first consider the problem for patterns with full vertical navigation, that is, with $\downarrow$ and $\downarrow^+$ edges.

▶ **Theorem 4.** *There is an automaton $\mathcal{A}$ such that* CONS$_\mathcal{A}$ *is* NP*-complete. Moreover,* CONS$_\mathcal{A}$ *is* NP*-hard already for patterns with at most two occurrences of $\downarrow^+$ per branch.*

Proof. The NP upper bound can be proved by a standard guess and check technique [2]. The rest of this proof is devoted to showing that the problem is NP-hard.

Consider the language K defined in Figure 1. It is straightforward to construct an automaton recognizing K. We claim that for any automaton $\mathcal{A}$ recognizing K, CONS$_\mathcal{A}$ is NP-hard (even for patterns with at most two occurrences of $\downarrow^+$ per branch).

We reduce from CNF-SAT. Let $\varphi = c_1 \wedge c_2 \wedge \dots \wedge c_m$ be a conjunction of clauses over variables $x_1, x_2, \dots, x_n$. We build a pattern π_φ such that the formula φ is satisfiable if and only if the pattern π_φ is satisfiable in a tree T from K.

The pattern π_φ can be decomposed in two parts. One part ensures that the tree T represents precisely the formula φ. The rest of the pattern represents a valuation of the variables $x_1, x_2, \dots, x_n$ and the proof that this valuation satisfies the formula φ. The idea of the encoding of the formula into a tree T from K is to associate each variable x_i with an

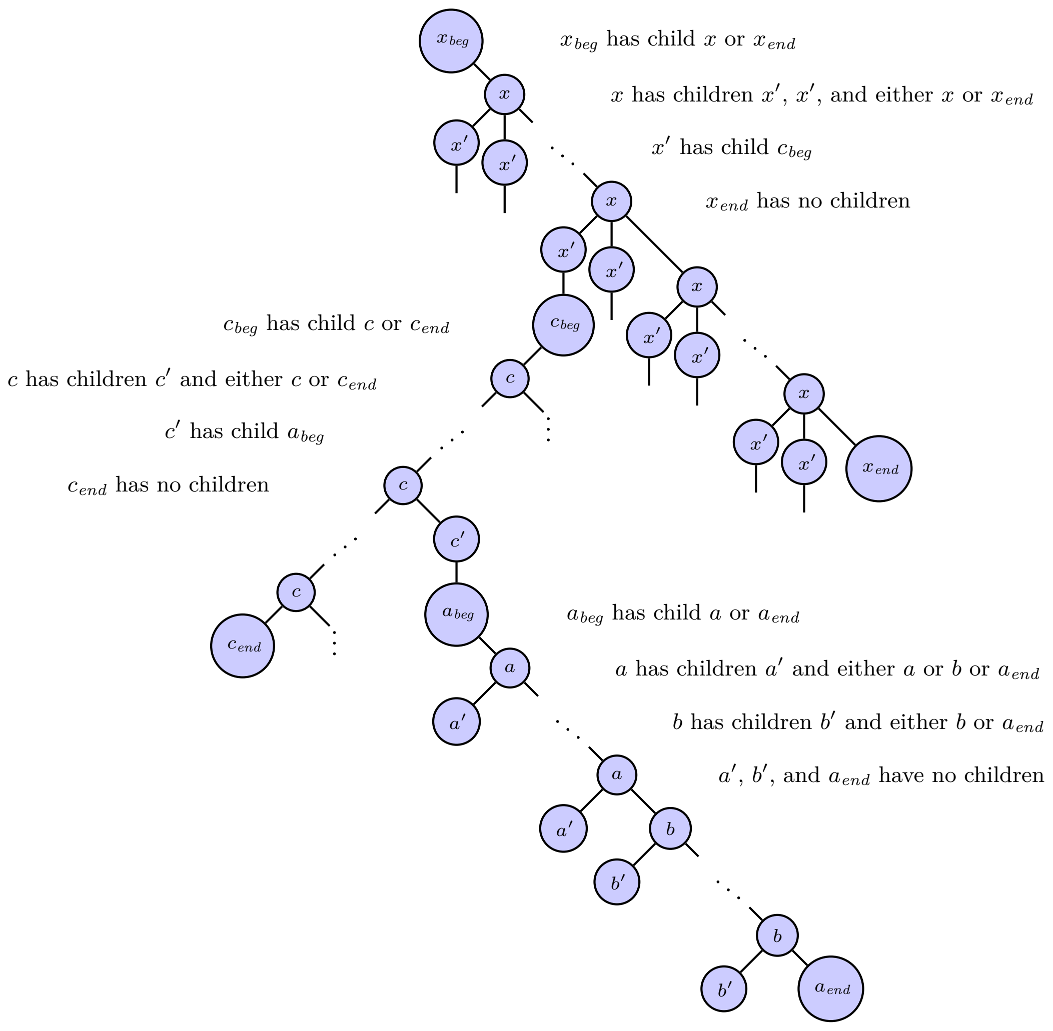

x_{beg} has child x or x_{end}
x has children x', x', and either x or x_{end}
x' has child c_{beg}
x_{end} has no children
c_{beg} has child c or c_{end}
c has children c' and either c or c_{end}
c' has child a_{beg}
c_{end} has no children
a_{beg} has child a or a_{end}
a has children a' and either a or b or a_{end}
b has children b' and either b or a_{end}
a', b', and a_{end} have no children

Figure 1 The tree language recognized by the automaton $\mathcal{A}$ used in the reduction of CNF-SAT to CONS$_{\mathcal{A}}$.

x node and encode in the two corresponding x'-rooted subtrees two lists of clauses: those satisfied when x_i is true, and those satisfied when it is false.

The full pattern π_φ is given in Figure 2. Notice that subpattern F^i_j depends on whether literal $\bar{x}_i$ occurs in the clause c_j or not; subpattern T^i_j is defined analogously, with literal $\bar{x}_i$ replaced with x_i. Let π'_φ be the pattern obtained from π_φ by removing subpatterns V^i and C^j for all i and j. In other words, we keep the blue nodes, but remove the green and red nodes. Observe that whenever π'_φ is matched in a tree $T \in K$, the subpatterns T^i and F^i must be matched at the grandchildren of the ith x node. Indeed, for T^n and F^n there is no choice. Consequently, since the matching must be injective, for T^{n-1} and F^{n-1} there is no choice either, etc. A similar argument applies to the subpatterns T^i_j and F^i_j. This implies that (up to the ordering of x' siblings) there is exactly one tree in K satisfying π'_φ: the tree T_φ obtained from π'_φ by filling in the missing nodes with labels x', c', a', b'. Moreover, there is exactly one injective homomorphism from π'_φ to T_φ, that is the one induced by the construction of T_φ.

Intuitively, the subpattern T^i lists the clauses of φ that are made true by setting x_i to

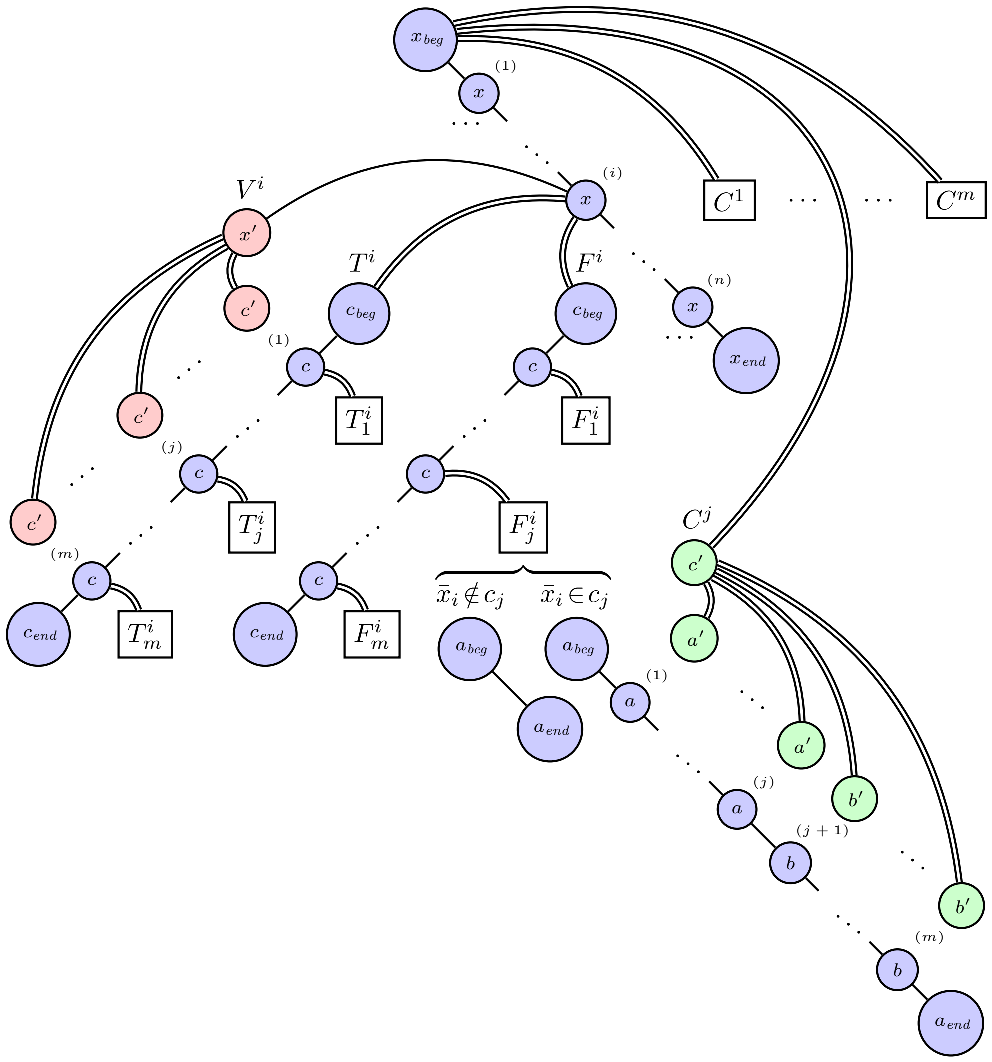


Figure 2 The pattern encoding a CNF formula $c_1 \wedge c_2 \wedge \cdots \wedge c_m$ over variables $x_1, x_2, \ldots, x_n$. Single and double lines represent child and descendant edges, respectively.

true, and F^i lists the ones made true by setting x_i to false. Whether clause c_j is true or not is encoded by subpatterns T^i_j and F^i_j: a sequence of j labels a and $m-j$ labels b is inserted between a_{beg} and a_{end} if and only if clause c_j is made true.

It remains to show that this homomorphism can be extended to the full pattern π_φ if and only if φ is satisfiable. There are two ways of matching V^i in T_φ: at the parent of the image of T^i or at the parent of the image of F^i. In either case, the matching uses all c' nodes in the corresponding subtree, while the nodes in the other subtree remain unused. Thus, choosing T^i should be interpreted as setting x_i to false, since c' nodes under F^i remain unused, and choosing F^i as setting x_i to true, since c' nodes under T_i remain unused. When all subpatterns V^i have been matched, subpattern C^j can be matched if and only if the associated valuation makes clause c_j true.

It follows that $T_\varphi \models \pi_\varphi$ if and only if there exists a valuation of the variables $x_1, x_2, \ldots, x_n$ that makes true every clause of φ. ◀

4 Descendant-only patterns

In the previous section we have proved that $\textsc{Cons}_{\mathcal{A}}$ is NP-complete in general. We know that the problem is tractable for some restricted classes of patterns such as patterns using only child relation [2] or the class considered by Kopczynski [10]. In this section, we prove that $\textsc{Cons}_{\mathcal{A}}$ is also tractable for tree patterns that only use the descendant relation.

▶ **Theorem 5.** *Let $\mathcal{A}$ be a fixed tree automaton. Then $\textsc{Cons}_{\mathcal{A}}$ is solvable in* PTIME *for $\downarrow^+$-only tree patterns.*

The key argument to prove Theorem 5 is that consistency of a descendant-only tree pattern with respect to an automaton $\mathcal{A}$ can be reduced to membership of the underlying tree of the pattern in a regular tree language that depends only on $\mathcal{A}$. When the automaton $\mathcal{A}$ is fixed, the latter can be checked in time polynomial in the size of π. This stronger result is proved in Lemma 14. The remaining of the section is dedicated to a fine analysis of descendant-only tree patterns together with a tree automaton, providing the tools needed to state and prove this lemma.

Our goal is to build concise representations of trees in $L(\mathcal{A})$ that satisfy some descendant-only pattern π, in such a way that the size of these representations does not depend on π. The first step is to omit nodes that are not used to satisfy π. The notion of *descendant count* introduced in Definition 6 provides a concise way to represent the set of possible frontiers that are reachable starting from a given label-state pair in a run of $\mathcal{A}$.

▶ **Definition 6.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. A *count* for $\mathcal{A}$ is a function $\alpha : \Sigma \times Q \to \overline{\mathbb{N}}$, where $\overline{\mathbb{N}} = \mathbb{N} \cup \{*\}$, with the natural order extended with $i \leq *$ for all $i \in \mathbb{N}$. We say that count α is smaller than count β if $\alpha(a,q) \leq \beta(a,q)$ for all pairs $(a,q) \in \Sigma \times Q$.

We say that a count α is *realized* at (a,q) if for all $n \in \mathbb{N}$, there exists a tree T, a run ρ of $\mathcal{A}$ on T, and a frontier w in T such that

- the root v of T has label a and $\rho(v) = q$;
- for all $(a',q') \in \Sigma \times Q$ such that $\alpha(a',q') \in \mathbb{N}$, w contains at least $\alpha(a',q')$ nodes v with label a' and such that $\rho(v) = q'$;
- for all $(a',q') \in \Sigma \times Q$ such that $\alpha(a',q') = *$, w contains at least n nodes v with label a' and such that $\rho(v) = q'$.

Finally, given $(a,q) \in \Sigma \times Q$, the *descendant count* of a and q, denoted by $\mathrm{DC}_{\mathcal{A}}(a,q)$, is defined as the set of all maximal counts for $\mathcal{A}$ that are realized at (a,q).

▶ Remark. The sets $\mathrm{DC}_{\mathcal{A}}(a,q)$ are finite and can be computed. Indeed, we can easily compute a context-free grammar recognizing the set $\mathrm{Fr}_{\mathcal{A}}(a,q) \subseteq (\Sigma \times Q)^*$ of sequences of letter-state pairs yielded by the frontiers occurring in the definition of $\mathrm{DC}_{\mathcal{A}}(a,q)$. As $\mathrm{Fr}_{\mathcal{A}}(q,a)$ is closed under subsequences, its Parikh image is a (finite) union of linear sets of the form $\{\beta \in \mathbb{N}^{\Sigma \times Q} \mid \beta \leq \alpha\}$, where α is a count. Since a semilinear representation of the Parikh image of a context-free language can be computed effectively, the involved counts α can be deduced as well. $\mathrm{DC}_{\mathcal{A}}(a,q)$ consists of the maximal ones among them.

Using descendant counts, we define the notion of skeleton for a tree automaton $\mathcal{A}$ which can be seen as a sparse representation of a tree in $L(\mathcal{A})$, where some nodes are omitted. We show that if a tree pattern π is satisfied by a skeleton s for $\mathcal{A}$, then it is consistent with $\mathcal{A}$.

▶ **Definition 7.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. A *skeleton* s for $\mathcal{A}$ is a tree whose nodes carry a label from $\Sigma \times Q$ and can optionally be flagged as starred. Additionally, for each node v of s with label (a,q), there exists $\alpha \in \mathrm{DC}_{\mathcal{A}}(a,q)$ such that for all (a',q')

- if $\alpha(a', q') \in \mathbb{N}$, then v has at most $\alpha(a', q')$ children with label (a', q'), all non-starred;
- if $\alpha(a', q') = *$, then v has an arbitrary number of children with label (a', q'), all starred;
- if v is the root, then v is not starred, and q is productive.

We say that s *satisfies* a $\downarrow^+$-only tree pattern π if the underlying tree of s satisfies π.

Descendant counts are used to build skeletons and ensure that each level of the skeleton is consistent with $\mathcal{A}$ and can indeed be simulated by a tree in $L(\mathcal{A})$. This is more precisely shown in the following lemma, where we prove that, starting from a skeleton s, we can build a tree T in $L(\mathcal{A})$ that features the same nodes as s, arranged in the same descendant order.

▶ **Lemma 8.** *Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton and s be a skeleton for $\mathcal{A}$. Then there exists a tree T, a run ρ of $\mathcal{A}$ on T and an injective mapping $i : nodes_s \to nodes_T$ such that, for all nodes u, v of s,*

- *if $lab_s(u) = (a, q)$, then $lab_T(i(u)) = a$ and $\rho(i(u)) = q$;*
- *if $u \downarrow v$ in s, then $i(u) \downarrow^+ i(v)$ in T;*
- *if u is the root of s, then $i(u)$ is the root of T.*

Proof. We prove this by induction on the structure of s.

Assume that s consists of a single node u with label (a, q). By Definition 7, there exists a count $\alpha \in \mathrm{DC}_{\mathcal{A}}(a, q)$. Since α is realized at (a, q), the tree T of Definition 6 satisfies the requirements of the lemma.

Assume that u is the root of s, with children $s_1, \ldots, s_n$. Let (a, q) be the label of u, and (a_i, q_i) be the label of the root of s_i. Then, by definition of s, there exists a count α that is realized at (a, q) and fits the definition of s at u. Then, by Definition 6, there exists a tree T and a run ρ on T such that T has root v with $lab_T(v) = a$, $\rho(v) = q$, and with some frontier $v_1 \ldots v_n$ with $lab_T(v_i) = a_i$ and $\rho(v_i) = q_i$. Then we can build from T the required tree by replacing the nodes of this frontier with the trees $T_1, \ldots, T_n$ produced by the induction hypothesis applied to $s_1, \ldots, s_n$. ◀

Since the root of a skeleton is always labeled by a productive state and our patterns only use $\downarrow^+$, Lemma 8 implies the following result.

▶ **Corollary 9.** *Let $\mathcal{A}$ be an automaton, s a skeleton for $\mathcal{A}$ and π a $\downarrow^+$-only pattern. If a skeleton s satisfies π, then there exists a tree $T \in L(\mathcal{A})$ that satisfies π.*

Note that, even though skeletons can be sparser than trees, there is still an infinite number of them. We show that we can represent all skeletons considering only the finite set of *reduced skeletons*.

▶ **Definition 10.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. A *reduced skeleton* s for $\mathcal{A}$ is a skeleton that additionally satisfies the following two properties:

- each pair label-flag appears at most once in each branch of s;
- each node of s has at most one starred child of each label.

Note that the number of reduced skeletons is finite for any automaton $\mathcal{A}$. Indeed, reduced skeletons are both bounded in depth, as there are a finite number of labels and they are not allowed to repeat along a branch, and in width, since the maximum number of non-starred children of any given label is bounded by the largest value different from $*$ taken by any of the counts in $\bigcup_{(a,q)\in\Sigma\times Q} \mathrm{DC}_{\mathcal{A}}(a, q)$.

Intuitively these skeletons correspond to minimal ones and can be obtained by pruning long branches and large siblings sets in some larger skeleton. Moreover, reduced skeletons contain enough information to recover the whole skeletons, by means of the horizontal and vertical pumping properties of tree automata.

▶ **Definition 11.** Skeleton s reduces to skeleton s' if s' can be obtained from s by applying a (possibly empty) sequence of the following reductions:

(H) Remove any starred node of s that has the same label as some of its starred siblings.
(V) Assume that a node u of s and its descendant v carry the same labels and flags. Then reduce s to a skeleton obtained by replacing in s the subtree s_u with s_v.

We write red(s) for the set of skeletons to which s reduces, that cannot be further reduced.

Note that the label and flag of the root of s are preserved by both reduction steps. Also, if s reduces to s', and s is a skeleton for $\mathcal{A}$ then s' is also a skeleton for $\mathcal{A}$. Moreover if s cannot be reduced by either (H) or (V), then s is a reduced skeleton. This implies that red(s) is the set of all reduced skeletons s' such that s reduces to s'.

The reductions (H) and (V) give a way to simplify a skeleton. The final ingredient we need is a way of combining skeletons without losing information. To this end we define the notion of *injection* of a skeleton into another. Intuitively an injection of s_2 into s_1 can be viewed as a skeleton s expanding s_1 such that s_2 can be matched disjointly from s_1 into s.

▶ **Definition 12.** Let s, s_1 and s_2 be skeletons. Then s is an *injection* of s_2 into s_1 if there exists two injective mappings $i_1 : \mathit{nodes}_{s_1} \to \mathit{nodes}_s$ and $i_2 : \mathit{nodes}_{s_2} \to \mathit{nodes}_s$ such that

- if u is the root of s_1, then $i_1(u)$ is the root of s;
- the images of i_1 and i_2 are disjoint;
- mappings i_1 and i_2 preserve labels and flags as well as descendant relation.

▶ Remark. Note that if s_1 satisfies a pattern π_1 and s_2 satisfies a pattern π_2, then any injection of s_2 into s_1 satisfies π_1 and π_2 *simultaneously*, that is, we can match π_1 and π_2 in such a way that their images are disjoint.

We are now ready to define the tree automaton $\mathcal{A}_\Pi$ and prove that it recognizes the set of all descendant-only tree patterns that are consistent with a given tree automaton $\mathcal{A}$. As explained in the beginning of the section Theorem 5 follows directly from this result.

▶ **Definition 13.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. Then we define the *pattern automaton* $\mathcal{A}_\Pi = (\Sigma, Q_\Pi, \delta_\Pi, F_\Pi)$ of $\mathcal{A}$ as follows.

- $Q_\Pi = F_\Pi$ is the set of all reduced skeletons for $\mathcal{A}$.
- Let s be a reduced skeleton for $\mathcal{A}$ and $a \in \Sigma$, then $s_1 \dots s_n \in \delta(s, a)$ if and only if there exist skeletons $t_0, \dots, t_n$ such that
  - t_0 is the root of s and is labeled (a, q) for some q;
  - for all $i > 0$, there exists an injection of s_i into t_{i-1} that reduces to t_i;
  - $t_n = s$ (or $t_0 = s$ if $s_1 \dots s_n$ is ε.)

It is easy to check that $\mathcal{A}_\Pi$ is a properly defined tree automaton. Indeed, the three properties defining $\delta(s, a)$ actually define the initial states, transitions and final states of a finite automaton, hence $\delta(s, a)$ is regular.

▶ **Lemma 14.** *Let $\mathcal{A}$ be a tree automaton and π be a $\downarrow^+$-only tree pattern. Then π is consistent with respect to $\mathcal{A}$ if and only if $\pi \in L(\mathcal{A}_\Pi)$.*

Proof. ($\Rightarrow$) Assume that π is consistent with respect to $\mathcal{A}$. We want to exhibit an accepting run ρ of $\mathcal{A}_\Pi$ on π.

Let $T \in L(\mathcal{A})$ such that $T \models \pi$, which means that there is an injective homomorphism h from π to T. Let μ be an accepting run of $\mathcal{A}$ on T. Combining, the tree T, the run μ and the pattern π, we build a skeleton s as follows:

- the nodes of s correspond to the nodes in $h(\pi)$;
- for each node v of π with label a, the corresponding node in s has label $(a, \mu(h(v)))$;
- the father of a node v in s is its closest ancestor in T that also belongs to $h(\pi)$;
- for each node v of s of label (a, q), choose $\alpha \in \mathrm{DC}_{\mathcal{A}}(a, q)$ such that the number of v's children of label (a', q') is at most $\alpha(a', q')$, and flag the children as starred accordingly.

Regardless of the choices of α, the resulting s is indeed a properly defined skeleton for $\mathcal{A}$, as T and μ witness all the required descendant counts. Note also that s satisfies π through the same injective homomorphism h.

For all nodes v of π, we define π_v as the subpattern of π rooted at v. For V, a subset of the set of nodes of π, we deduce s^0_V from s by keeping only the least common ancestor of nodes in V as well as all the nodes of s that appear in $h(\pi_v)$ for all $v \in V$, and linking nodes to their closest ancestor, as it is done for s. For s^0_V to be a proper skeleton, we also unflag its root in case it is flagged as starred. We also define s_V as any skeleton arbitrarily chosen in $\mathrm{red}(s^0_V)$. If V consists of a single node v, we simply write s^0_v and s_v.

We are now ready to exhibit an accepting run ρ of $\mathcal{A}_\Pi$ on π. For each node v of π, we define $\rho(v) = s_v$. It remains to show that ρ is a properly defined run of $\mathcal{A}_\Pi$; it will immediately be accepting, as all states of $\mathcal{A}_\Pi$ are final. We show by induction on the structure of π that, for all nodes v of π, the partial run defined by ρ on π_v is a correct run for $\mathcal{A}_\Pi$.

Let v be a leaf node of π with label a. Then s^0_v is a skeleton consisting of a single node labeled (a, q) for some q, and is thus reduced. Hence, $\rho(v) = s_v = s^0_v$, $\varepsilon \in \delta_\Pi(a, s_v)$ and ρ is a properly defined run on π_v.

Let v be an internal node of π with label a. Let $u_1, \ldots, u_n$ be the children of v. By the induction hypothesis, we know that ρ is a properly defined run on all π_{u_i}. Let t_0 be the root of s_v. As $h(v) = t_0$, then t_0 has label (a, q) for some q. For all $i > 0$, we define $V_i = \{u_1, \ldots, u_i\}$ and $t_i = s_{V_i}$. Then, this sequence of skeletons satisfies the definition of $\mathcal{A}_\Pi$. The injection of s_{u_i} into t_{i-1} is simply $s^0_{V_i}$, which reduces to t_i by definition. Hence, ρ is a properly defined run on π_v.

($\Leftarrow$) Assume that $\pi \in L(\mathcal{A}_\Pi)$. Let ρ be an accepting run of $\mathcal{A}_\Pi$ on π. For each node v of π, we define π_v as the subpattern of π rooted at v. We now prove by induction on the structure of π that, for all nodes v of π, there exists a skeleton s that satisfies π_v and that reduces to $\rho(v)$.

Let v be a leaf node of π with label a. By definition of $\mathcal{A}_\Pi$, the reduced skeleton $\rho(v)$ is a single node labeled (a, q) for some q. Then $\rho(v)$ satisfies π_v and is already reduced. Hence, we can choose $s = \rho(v)$.

Let v be an internal node of π labeled a. Assume that v has only two children, v_1 and v_2, as other cases are similar. Let u be the root of $\rho(v)$. By definition of $\mathcal{A}_\Pi$, there is an injection t of $\rho(v_1)$ and $\rho(v_2)$ into u that reduces to $\rho(v)$. By induction, there are two skeletons s_1 and s_2 that respectively reduce to $\rho(v_1)$ and $\rho(v_2)$ and respectively satisfy π_{v_1} and π_{v_2}.

We can build from t a skeleton s by reverting in t all the reductions steps that are used to reduce each s_i to $\rho(v_i)$, as well as adding enough copies of starred nodes of t so that s is an injection of s_1 and s_2 into u. Thus, s satisfies both π_{v_1} and π_{v_2} simultaneously without using the root node. Moreover, it is easy to check that s reduces to $\rho(v)$, since all new nodes can simply be removed by reductions steps. Let h be an injective homomorphism that witnesses the fact that s satisfies π_{v_1} and π_{v_2} simultaneously and without using the root node. Then we can extend h by mapping v to u. This extended mapping witnesses the fact that s satisfies π_v, as u has label (a, q) for some q, since it is the root of $\rho(v)$.

By applying this induction to the root v of π, we deduce that there exists a skeleton s that reduces to $\rho(v)$ and satisfies π. We conclude using Lemma 8 and Corollary 9. ◀

5 Extending the pattern language

In this section we briefly discuss possible extensions of the pattern language. Let us first observe that we can add wildcard to our language for free, that is, we can costlessly allow nodes in patterns that do not have a specified label and can match a tree node with any label. Indeed, our automaton can simply guess the label for each processed wildcard, and then proceed as before.

A more interesting extension is to add horizontal relations. Patterns with horizontal relations are defined just like $\{\downarrow, \downarrow^+\}$-patterns we have seen so far, except that they have two additional kinds of edges, denoted by $\rightarrow$ and $\rightarrow^+$, and interpreted respectively as the next sibling and the following sibling.

As soon as we add the next sibling relation, the consistency problem becomes NP-hard. A reduction can be obtained via a simple modification of the one in Theorem 4. Specifically, it suffices to modify the encoding so that the x nodes, c nodes, and a and b nodes are arranged horizontally, rather than vertically. After this modification the pattern in Figure 2 only uses child relation between x and x' nodes. Given that the only descendants of any x node that have label x' are its children, we can replace the child relation with the descendant relation.

▶ **Theorem 15.** *There is an automaton $\mathcal{A}$ s. t.* $\textsc{Cons}_\mathcal{A}$ *is* NP*-complete for $\{\downarrow^+, \rightarrow\}$-patterns.*

When only the following sibling is added, we can get a polynomial algorithm.

▶ **Theorem 16.** *For each automaton $\mathcal{A}$,* $\textsc{Cons}_\mathcal{A}$ *is in* PTIME *for $\{\downarrow^+, \rightarrow^+\}$-patterns.*

In fact, we can again construct a tree automaton recognizing $\{\downarrow^+, \rightarrow^+\}$-patterns consistent with an automaton $\mathcal{A}$. In the following, we explain the main ideas of this construction.

We first explain how to extend the notion of skeleton. Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. We assume that horizontal languages in the automaton are given in disjunctive normal form, that is, for each $(a, q) \in \Sigma \times Q$, the language $\delta(a, q)$ is given by a disjunction of disjunction-free regular expressions. We shall refer to these disjunction-free expressions as *clauses* of $\delta(a, q)$. Note that turning a regular expression into this form usually involves an exponential blow-up, but since the automaton is considered to be fixed, this does not change the complexity bound. In the definition below, a letter-state pair (a, q) is *reachable* if there exists a tree T with label a in the root and a run over T that assigns state q to the root. A state q is reachable if there exists a run on any tree that assigns q to the root. Without loss of generality we can assume that all states of $\mathcal{A}$ are reachable.

▶ **Definition 17.** A $\{\downarrow^+, \rightarrow^+\}$-*skeleton* (in this section, just *skeleton*) for an automaton $\mathcal{A} = (\Sigma, Q, \delta, F)$ is a forest labelled with disjunction-free regular expressions over reachable letter-state pairs from $\Sigma \times Q$ such that

- each label is either a single letter-state pair (non-starred node) or a disjunction-free regular expression of the form e^* (starred node);
- starred nodes have no children;
- for each node of label (a, q) the concatenation of the labels of its children forms a disjunction-free regular expression $w_1 u_1 (e_1)^* v_1 w_2 u_2 (e_2)^* v_2 \ldots w_n u_n (e_n)^* v_n w_{n+1}$ such that u_i, v_i are generated by e_i and the projection over Q of $w_1(e_1)^* w_2(e_2)^* \ldots w_n(e_n)^* w_{n+1}$ is a clause of $\delta(a, q)$;
- similarly for the concatenation of labels of the roots, except that the projection over Q of $w_1(e_1)^* w_2(e_2)^* \ldots w_n(e_n)^* w_{n+1}$ is a *suffix* of a clause of $\delta(a', q')$ for some productive q'.

Additionally, non-starred nodes can be flagged as used.

A skeleton is *reduced* if no letter-state pair repeats on a branch, and the words u_i and v_i in the definition above are all empty. A reduced skeleton has its branching bounded by the size of the clauses of the horizontal languages (which are polynomial in the original representation of the languages), and its height bounded by the number of states of the automaton $\mathcal{A}$. Hence, the set of reduced skeletons is finite and each of them is of size at most exponential in the size of $\mathcal{A}$.

Like for $\downarrow^+$-skeletons, we can reduce skeleton s by repeatedly applying the following rules

(H) remove any non-starred node (together with its subtree) whose label occurs in the regular expression e^* labelling its next or previous sibling;

(V) if u and its descendant v are non-starred and carry the same label, then the subtree rooted at u (excluding u) can be replaced by the subtree rooted at v (excluding v).

The automaton recognizing consistent patterns essentially proceeds like before: it assigns reduced skeletons to nodes of the pattern π in a bottom-up fashion, ensuring that they are consistent with each other. More precisely, a node v gets a skeleton that summarizes a way to satisfy the subpattern of π rooted at v. Note that in this subpattern some nodes are connected to v via $\downarrow^+$-edges, and others via $\to^+$-edges. Thus, the subpattern talks about a certain subforest, which explains why our skeletons are forests. We always assume that v is mapped to the first root of the skeleton.

Suppose that we want to assign a reduced skeleton to a node v. First, we guess a reduced skeleton for a single-node pattern consisting of v alone. This skeleton has at most one used node. Next, we aggregate it with the skeletons assigned to v's children, one by one, using appropriately adjusted injections. Since v's children are now connected to v via $\downarrow^+$ or $\to^+$, we need two variants of the notion. In both variants, we add to Definition 12 an item guaranteeing preservation of the sibling order: if $v \to^+ v'$ in s_k, then $i_k(v) \to^+ i_k(v')$ in s. In the variant for $\downarrow^+$, we require that the first root of the second skeleton is mapped to a descendant of the first root of s, and in the variant for $\to^+$, it is mapped into a following sibling of the first root of s.

We close this section by commenting that the reasoning above could be extended to cover limited use of child and next-sibling relations: it can be done for patterns, where the maximal length of paths that do not use $\downarrow^+$-edge is bounded.

6 Conclusions

We have shown that under injective semantics, the consistency problem for tree patterns with respect to a fixed automaton is NP-complete by showing the problem to be NP-hard already for child/descendant patterns with at most two descendant edges per branch. This closes an open problem from [2]. Moreover our result is tight with respect to the result of Kopczynski [10], showing tractability for patterns with at most one descendant per branch.

On the positive side, we have provided a polynomial time algorithm in the case of descendant-only tree patterns. The key ingredient is to show that the set of all patterns that are consistent with a given tree automaton $\mathcal{A}$ is a regular tree language. This language only depends on $\mathcal{A}$ and we can effectively construct a tree automaton $\mathcal{A}_\Pi$ recognizing it. Hence, consistency is equivalent to testing whether the pattern belongs to this language, which can be done in polynomial time. Thus, our algorithm is not only polynomial for fixed $\mathcal{A}$, but also fixed-parameter tractable with the size of $\mathcal{A}$ as the parameter.

The involved constant is essentially the size of the automaton $\mathcal{A}_\Pi$, which is double exponential in the size of $\mathcal{A}$. This may seem suboptimal, since the problem is known to be in

NP even when $\mathcal{A}$ is a part of the input. However, while we are guaranteed to find a witness polynomial in the size of the pattern and the automaton, it may be arbitrarily large with respect to the automaton itself. It happens so that these witnesses can be summarized as objects exponential in the size of the automaton (double exponential complexity comes from handling sets of such summaries), but we can see no way to do better than exponential.

We have also examined patterns with additional features: wildcard can be added effortlessly, but horizontal relations pose more problems. We adapted our techniques to show that one can combine descendant and following-sibling without losing tractability, but combining descendant with next-sibling makes the problem NP-complete (for some automata).

Given that without descendant the problem is known to be tractable [2], this charts out completely the tractability frontier for the consistency of injective tree patterns. A question we find interesting and challenging is which of the tractability results can be extended to patterns that are DAGs, rather then trees. For instance, what is the complexity of the consistency problem for descendant-only DAG patterns?

Acknowledgements. We thank the anonymous referees for their comments. The third author was supported by Poland's National Science Centre grant no. UMO-2013/11/D/ST6/03075.

References

1 Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

2 Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *J. ACM*, 58(1):4, 2010.

3 Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008.

4 Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Semi-structured data with constraints and incomplete information. In *Description Logics*, 1998.

5 Claire David. Complexity of data tree patterns over XML documents. In *MFCS*, pages 278–289, 2008.

6 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

7 Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer, 1991.

8 Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

9 Yaron Kanza, Werner Nutt, and Yehoshua Sagiv. Querying incomplete information in semistructured data. *J. Comput. Syst. Sci.*, 64(3):655–693, 2002.

10 Eryk Kopczynski. Trees in trees: Is the incomplete information about a tree consistent? In *CSL*, pages 367–380, 2011.

11 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

12 Maarten Marx. XPath with conditional axis relations. In *EDBT*, pages 477–494, 2004.

13 Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.

14 Frank Neven and Thomas Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Log. Meth. Comput. Sci.*, 2(3), 2006.

15 Peter T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT*, pages 297–311, 2003.

Asymptotically Optimal Encodings for Range Selection*

Gonzalo Navarro[1], **Rajeev Raman**[2], **and Srinivasa Rao Satti**[3]

1 **Department of Computer Science, University of Chile, Chile**
`gnavarro@dcc.uchile.cl`

2 **Department of Computer Science, University of Leicester, UK**
`r.raman@leicester.ac.uk`

3 **School of Computer Science & Engg, Seoul National University, S. Korea**
`ssrao@cse.snu.ac.kr`

Abstract

We consider the problem of preprocessing an array $A[1..n]$ to answer *range selection* and *range top-k* queries. Given a query interval $[i..j]$ and a value k, the former query asks for the position of the kth largest value in $A[i..j]$, whereas the latter asks for the positions of all the k largest values in $A[i..j]$. We consider the *encoding* version of the problem, where A is not available at query time, and an upper bound κ on k, the rank that is to be selected, is given at construction time. We obtain data structures with asymptotically optimal size and query time on a RAM model with word size $\Theta(\lg n)$: our structures use $O(n \lg \kappa)$ bits and answer range selection queries in time $O(1 + \lg k / \lg \lg n)$ and range top-k queries in time $O(k)$, for any $k \leq \kappa$.

1998 ACM Subject Classification F.2.2. Nonnumerical Algorithms and Problems, E.2 Data Storage Representations, E.4 Coding and Information Theory

Keywords and phrases Data Structures, Order Statistics, Succinct Data Structures, Space-efficient Data Structures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.291

1 Introduction

We consider the problem of preprocessing an array $A[1..n]$ over a totally ordered universe, so that the following queries can be efficiently answered:

- Range selection: $\mathsf{select}(i, j, k)$ returns the position of the kth largest element in $A[i..j]$.
- Range top-k: $\mathsf{top}(i, j, k)$ returns the positions of the k largest elements in $A[i..j]$.

We can assume that A is a permutation of $[n]$, since replacing each element $A[i]$ by its rank in A yields correct answers to those queries. The range selection problem has received a lot of interest in recent years [4, 3, 13, 5]. Following a series of earlier papers, Brodal and Jørgensen [4] presented a structure using linear space and $O(\lg n / \lg \lg n)$ time, for any k given at query time. The model used for this result, as well as the other results in this paper, is the *word RAM* model with word size $w = \Theta(\log n)$ bits. Jørgensen and Larsen [13] improved the time to $O(\lg k / \lg \lg n + \lg \lg n)$, still within linear space, and proved that $\Omega(\lg k / \lg \lg n)$ time is needed when using $n \lg^{O(1)} n$ space. Finally, Chan and Wilkinson [5]

* Navarro funded in part by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F; Satti partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant number 2012-0008241).

matched this lower bound, obtaining $O(1 + \lg k/\lg\lg n)$ time using linear space[1]. This result implies, via a reduction first observed in [4], an optimal $O(k)$-time solution to the range top-k problem as well.

In this paper, we are interested in the *encoding model*, where the array A is not available at query time, and therefore the data structure must contain enough information to answer queries by itself. One can always use a non-encoding data structure such as that of Chan and Wilkinson [5], on a copy A' of A, and thus trivially avoid access to A at query time. This yields an encoding that uses $O(n)$ words, or $O(n \log n)$ bits, and has time equal to that of the best non-encoding data structure. We aim to find non-trivial encodings of size $o(n \log n)$ bits (from which, of course, it is not possible to recover the sorted permutation, but one can still answer any select query).

Existing non-trivial solutions for this problem in the encoding model are as follows. In the case $k = 1$, both queries boil down to the well-known *range maximum query (RMQ)*, which can be answered in constant time and $2n + o(n)$ bits, matching the lower bound of $2n - O(\lg n)$ bits to within lower-order terms [9]. Note that the space usage is $O(n/\lg n)$ words, or sublinear. The case $k = 2$ was recently considered by Davoodi et al. [7]. Grossi et al. [11] considered encodings for general k, showing that $\Omega(n \lg k)$ bits are needed to encode answers to either selection or top-k queries. Therefore, interesting encodings can only exist if an upper bound κ on k is given at construction time—the so-called *κ-bounded rank* variant of this problem [13]. For general k, Grossi et al. [11] gave an asymptotically optimal-space and $O(1)$ time solution for the (much simpler) case where k is fixed at construction time and furthermore, only *one-sided* queries (i.e. query intervals of the form $A[1, j]$) are supported. Optimal-space encodings for the two-sided range selection problem can be obtained via encodings of the range top-k problem given by Grossi et al. [11] described below; these however have poor running times. Chan and Wilkinson gave a (bounded-rank) range selection encoding for general k that answers select queries in $O(1 + \lg k/\lg\lg n)$ time. Its space usage, however, is $O(n(\lg \kappa + \lg\lg n + (\lg n)/\kappa))$ bits, which is non-optimal.

In this paper we show that the same optimal time can be obtained in the encoding model, using asymptotically optimal space.

▶ **Theorem 1.** *Given an array $A[1..n]$ and a value κ, there is an encoding of A that uses $O(n \lg \kappa)$ bits and supports the query* select(i, j, k) *in $O(1 + \lg k/\lg\lg n)$ time for any $k \leq \kappa$.*

Furthermore, our development allows us to obtain asymptotically optimal time and space for the encoding range top-k problem.

▶ **Theorem 2.** *Given an array $A[1..n]$ and a value κ, there is an encoding of A that uses $O(n \lg \kappa)$ bits and supports the query* top(i, j, k) *in time $O(k)$, for any $k \leq \kappa$.*

Grossi et al. [11] gave a range top-k encoding using $O(n \lg \kappa)$ bits that answers top-k queries in $O(\kappa)$ time, for any $k \leq \kappa$. To achieve the optimal $O(k)$ time, they require $O(n \lg^2 \kappa)$ bits. Note that Grossi et al.'s result implies an optimal-space (bounded-rank) range selection encoding with running time $O(\kappa)$.

In general, the low space usage of encoding data structures is useful when the values in A themselves are uninteresting, and one just wants to query about their relative magnitudes. An example of range top-k queries used for autocompletion search is given by Grossi et

[1] Chan and Wilkinson claim a bound of $O(1 + \log_w k)$ for the "trans-dichotomous" model where the word size $w = \Omega(\log n)$; this is, however, based on an incorrect application [17] of a result of Grossi et al. [12], and the proof presented in [5] only yields a time bound of $O(1 + \log k/\log\log n)$.

al. [11]; the problem arises frequently in data and log mining applications as well. In addition, our result for range selection allows, for example, delivering the top-k results in sorted order. It is also useful for interfaces where, say, the top-k results are displayed and then, upon user request, the $(k+1)$th to $2k$th results are displayed, and so on. Even when A is needed, the sub-linear space usage of encoding data structures means that multiple copies of range selection data structures can be built over one copy of A, and still take less space than A (this trick is used already in the non-encoding result of [5]).

The next section gives some basic concepts and the roadmap of the paper.

2 Preliminaries

Grossi et al. [11] build their results on top of the *shallow cutting* technique [13, 5]. We revisit (a slight variant of) this construction, as we also build on it.

Let $A[1..n]$ be a permutation on $[n]$. Furthermore, consider each entry $A[i]$ as a point $(x, y) = (i, A[i])$, and set a parameter κ. A horizontal line sweeps the space $[1, n] \times [1, n]$ from $y = n$ to $y = 1$. The points hit are included in a single *root cell*, which spans a three-sided area called a *slab*, of the form $[1, n] \times [y, n]$, including all the points of the cell. Once we reach a point (x^*, y^*) that makes the root cell contain 2κ points, we *close* the cell and leave its final slab as $[1, n] \times [y^*, n]$. Then we create two *children cells* of κ points as follows. Let x_{split} be the κth x-coordinate in the root cell. This is called the *split point*. Then the new cells contain the points whose x-coordinates are $\le x_{\text{split}}$ and $> x_{\text{split}}$, respectively, and their initial slabs are thus $[1, x_{\text{split}}] \times [y^*, n]$ and $[x_{\text{split}} + 1, n] \times [y^*, n]$ (these will grow downwards as we continue with the sweeping process, independently on each cell). When those cells reach size 2κ, they are split again, and so on. A binary tree T_C is created to reflect the cell refinement process. The root cell is associated with the root node of T_C, the first two children cells to the left and right children of the root, and so on. The leaves of T_C are associated with the final cells, which have not been split and contain κ to $2\kappa - 1$ points (unless $n < \kappa$).

At any moment of the sweeping process, there is a sequence of split points $x_1, x_2, \ldots$, which grows as further cells are split. The current leaves of T_C cover an interval of x-coordinates $[x_i + 1, x_{i+1}]$ (we implicitly assume split points 0 and n at the extremes). When the next split occurs, within the cell covering interval $[x_i + 1, x_{i+1}]$, we split the cell into two new cells covering the x-coordinate intervals $[x_i + 1, x_{\text{split}}]$ and $[x_{\text{split}} + 1, x_{i+1}]$. We associate the *keys* $[x_i + 1, x_{\text{split}}]$ and $[x_{\text{split}} + 1, x_{i+1}]$ and the *extents* $[x_{i-1} + 1, x_{i+1}]$ and $[x_i + 1, x_{i+2}]$, respectively, with the two new cells. After the sweep finishes, the sequence of split points is of the form $0 = x_0 < x_1 < x_2 < \ldots < x_{n'} = n$. In the following, we will use x_i to refer to this final sequence of split points. Then we add n' further *keyless* cells with extents $[x_{i-1} + 1, x_{i+1}]$ for all $1 \le i \le n'$. Note that $\kappa \le x_{i+1} - x_i \le 2\kappa$ for all i (if $n \ge \kappa$).

This construction has useful properties [13]: (i) it creates $O(n') = O(n/\kappa)$ cells, each containing κ to 2κ points (if $n \ge \kappa$); (ii) if c is the cell of the highest (closest to the root) node $v \in T_C$ whose key is contained in a query range $[i..j]$, then $[i..j]$ is contained in the extent of c; and (iii) the top-κ values in $[i..j]$ belong to the union of the points in the 3 cells comprising the extent of c.

With these properties, Chan and Wilkinson [5] reduce the $O(\lg n / \lg\lg n)$ time of Brodal and Jørgensen [4] as follows. At each node $v \in T_C$, they store the structure of Brodal and Jørgensen for the array $A_v[1..O(\kappa)]$ of the y-coordinates of the points in the extent of v. Actually, they store in A_v the local permutation in $[O(\kappa)]$ induced by the relative ordering in A, thus A_v requires $O(\kappa \lg \kappa)$ bits in each v and $O(n \lg \kappa)$ bits in total. The structure for range selection also uses $O(\kappa \lg \kappa)$ bits and answers queries in time $O(1 + \lg_w \kappa)$. They also store an array $P_v[1..O(\kappa)]$, so that $P_v[i]$ is the position in $A[1..n]$ of the value stored in $A_v[i]$.

Property (iii) above implies that the kth largest element of $A[i..j]$, for any $k \leq \kappa$, is also the kth largest value in $A_v[l,r]$, where v is the node that corresponds to interval $[i..j]$ by property (ii) and $P_v[l-1] < i \leq j < P_v[r+1]$ are the elements in the extent of node v enclosing $[i..j]$ most tightly. Thus query $\mathsf{select}(i,j,k)$ on A is mapped to query $p = \mathsf{select}(l,r,k)$ on A_v. Once the local answer is found in $A_v[o]$, the global answer is $P_v[o]$. Chan and Wilkinson [5] manage to store all the P_v arrays in $O(n \lg(\kappa \lg n) + (n/\kappa) \lg n)$ bits, which gives $O(n \lg n)$ bits when added over a set of suitable κ values. This is linear space, but too large for an encoding.

Grossi et al. [11] use an $O(n')$-bit representation of the topology of T_C [16] that carries out a number of operations in constant time, plus a bit-vector of length n to mark the x_i values. With these and some additional structures of total size $O(n)$ bits, they show how to find the appropriate node $v \in T_C$, as well as the cell and extent limits, corresponding to a range $A[i..j]$, in constant time. They can also map between i and x_i, and compute the interval $[x_l, x_r]$ of splitting points contained in any node v, all in constant time.

In the sequel we build a space- and time-optimal encoding for range selection:

1. In Section 3 we provide constant-time access to any P_v using only $O(n \lg \kappa)$ bits in the encoding model. This yields an $O(\lg \kappa)$ time algorithm for range selection, as we can first find the node v in constant time, then binary search for l and r in P_v, then run the range selection query on A_v in time $O(1 + \lg \kappa / \lg \lg n)$, to finally return $P_v[o]$ in $O(1)$ time. This is obtained by a hierarchical marking of nodes plus a color-based encoding of the inheritance of points along cells in paths of unmarked nodes in T_C.
2. In Section 4 we address the bottleneck of the previous solution: we replace the binary search by fast predecessor queries on P_v, so as to obtain $O(1 + \lg \kappa / \lg \lg n)$ time. This is obtained by storing *succinct string B-trees* (succinct SB-trees) [12] on some nodes, which enable a denser marking, and searches on the color information along (now shorter) paths of unmarked nodes, using global precomputed tables.
3. In Section 5 we wrap up the results in order to prove Theorem 1. Then we show how to answer top-k queries by first finding the kth element in A_v and then using existing techniques [15] to collect all the values larger than the kth. This proves Theorem 2.

3 Constant-time Access to P_v

We describe a data structure that gives constant-time access to the values $P_v[1..O(\kappa)]$ in any node v.

3.1 Marking Nodes

Let $s(v)$ be the number of descendants of v in T_C. We define a decreasing sequence of sizes as follows: $t_0 = n'$ and $t_{\ell+1} = \lceil \lg t_\ell \rceil$, until reaching a z such that $t_z = 1$. Node v will be of *level* ℓ if $t_\ell^2 \leq s(v) < t_{\ell-1}^2$. For any $\ell \geq 1$, we mark a node $v \in T_C$ if it is of level ℓ and:

C1. it is a leaf or both its children are of level $> \ell$; or
C2. both its children are of level ℓ; or
C3. it is the root or its parent is of level $< \ell$.

▶ **Lemma 3.** *The number of marked nodes of level ℓ is $O(n'/t_\ell^2)$.*

Proof. The key property is that the descendants of v are of the same level of v or less. So nodes marked by C1 above cannot descend from each other, thus each such marked node has at least t_ℓ^2 descendants not shared with another. As T_C has at most $2n'$ nodes, there cannot be more than $2n'/t_\ell^2$ nodes marked by this condition. By the same key property, nodes marked by C2 form a binary tree whose leaves are those marked by C1, thus there are at most other $2n'/t_\ell^2$ nodes marked by C2. For C3, note that all unmarked nodes of level ℓ are in disjoint paths (otherwise the parent of two nodes of level ℓ would be marked by C2), and the path terminates in a node already marked by C1 or C2 (contrarily, a node of level ℓ marked by C3 must be a child of a node of level $< \ell$, and thus cannot descend from nodes of level ℓ, by the key property). Therefore, C3 marks the highest node of each such isolated path leading to a node marked by C1 or C2, and thus the number of nodes marked this way is limited by those marked by C1 or C2. ◀

3.2 Handling Marked Nodes

Marked nodes, across all the levels, are few enough to admit an essentially naive storage of the array P_v. If a marked node v represents a slab with left boundary $x_l + 1$, we store all its $P_v[o]$ values as the integers $P_v[o] - x_l$. As explained, from v we can determine x_l, and thus obtain $P_v[o]$ in constant time. Since a node of level ℓ contains less than $t_{\ell-1}^2$ descendants (leaves, in particular), its slab spans $O(t_{\ell-1}^2)$ consecutive split points x_i, and thus $O(\kappa\, t_{\ell-1}^2)$ positions in A. Thus, each such integer $P_v[o] - x_l$ can be represented using $\lg O(\kappa\, t_{\ell-1}^2) = O(t_\ell + \lg \kappa)$ bits. The second term adds up to $O(\kappa \lg \kappa)$ bits per node and $O(n \lg \kappa)$ overall. Since, by Lemma 3, there are $O(n'/t_\ell^2)$ marked nodes of level ℓ, the first term, $O(t_\ell)$, adds up to $O((n'/t_\ell^2) \cdot (\kappa\, t_\ell)) = O(n/t_\ell)$ bits over all marked nodes of level ℓ. Adding over all the levels ℓ we have $O(n) \sum_{\ell=0}^{z} 1/t_\ell$. Since $t_z = 1$ and $t_{\ell-1} > 2^{t_\ell - 1}$, it holds $t_{z-s} > 2^s$ for $s \geq 4$, and thus $O(n) \sum_{\ell=0}^{z} 1/t_\ell \leq O(n)(O(1) + \sum_{s \geq 0} 1/2^s) = O(n)$ bits overall.

3.3 Handling Unmarked Nodes

While the problem of supporting constant-time access to P_v is solved for marked nodes, T_C may have $\Theta(n')$ unmarked nodes. To deal with unmarked nodes, we first observe that an unmarked node v at level ℓ has exactly one level ℓ child and one child x at level $> \ell$ (otherwise v would be marked by C2). Furthermore, x is marked by C3. Finally, the marked parent of an unmarked level ℓ node must be the root or at level ℓ itself. Thus, as already observed, level ℓ unmarked nodes form disjoint paths in T_C, and all nodes adjacent to such a path are marked.

Now consider the points in slabs corresponding to unmarked nodes. When a cell is closed and split into two, the leftmost (rightmost) κ points in its slab become part of its left (right) child slab.

Thus, each child slab starts out with κ *inherited* points which are in common with its parent slab and κ further *original* points will be added to it before it is itself closed and split. For each point of node v, in x-coordinate order, we use a bit to specify if the point is inherited or original. Let $o_v[1..2\kappa]$ be this bit-vector.

Let π be a path of unmarked nodes of level ℓ, let u be the marked parent of the topmost unmarked node, and let v be an unmarked node in π. Each original point p of v must be an inherited point of some marked descendant v' that is adjacent to π (recall that v' represents all its points explicitly). Thus the coordinate of each such original point p can be specified by recording which marked descendant v' contains it, and the rank of p among the points of v'. Suppose that the j-th original point in v is in v's marked descendant at distance d_j along π. Then we write down the bit-string $b_v = \mathbf{1}^{d_1 - 1}\mathbf{0}\mathbf{1}^{d_2 - 1}\mathbf{0} \ldots \mathbf{1}^{d_\kappa - 1}\mathbf{0}$. We claim that, summed across all nodes v in the path π, this adds $2|\pi|\kappa$ bits: there are $|\pi|\kappa$ $\mathbf{0}$ bits, each $\mathbf{1}$ bit represents an inherited point in a slab on the path π, and there are $|\pi|\kappa$ inherited points in π. Thus, $\sum_{v \in T_C} |b_v| = O(n'\kappa) = O(n)$ bits. As explained, we also store $O(\lg \kappa)$ bits for each original point in v telling which rank to pick in the marked node, in an array r_v. This adds $O(n'\kappa \lg \kappa) = O(n \lg \kappa)$ bits, which completes the information necessary to identify any original point. Section 3.4 has the details of how to obtain the point value in $O(1)$ time.

Unfortunately, we cannot apply the same approach to the inherited points in v, as we cannot bound the size of the bit-strings as we did for b_v. For any inherited point p in v, we instead specify which ancestor of v on π has p as an original point (we specify u if this ancestor is outside π), and then retrieve the point as an original point in the ancestor. This is done by coding points using 4κ *colors.* Of these colors, 2κ are *original* colors and 2κ are *inherited* colors. For each original color g there is a corresponding inherited color g'. All the points in u are given arbitrary distinct original colors. Then we traverse the nodes v in π top to bottom. If point p in v is inherited (from its parent v'), we look at the color of p in v'. If p has an original color g in v', we give p color g' in v. Otherwise, if p is also inherited in v', having color g', it will also have color g' in v. On the other hand, if point p is original in v, we give it one of the currently unused original colors. Note that no colors g and g' can be present simultaneously in any v', thus writing g' in v unambiguously determines which color is inherited from v'. Then any other color g such that g' is not among the κ inherited colors of v can be used as an original color for v.

This scheme gives sufficient information to track the inheritance of points across π: when a new, original, point p appears in v, it is given an original color g. Then the point is inherited along the descendants of v as long as color g' exists below v. Thus, to find the appropriate ancestor of v that contains a given inherited point p of color g', as an original point, we concatenate all the colors on π into a string, and ask for the nearest preceding occurrence of color g. The path can be encoded in $O(|\pi|\kappa \lg \kappa)$ bits, which adds up to $O(n \lg \kappa)$ bits overall. The position of g in the nearest ancestor also tells which of the original points does p correspond to.

3.4 Technicalities

Let us fix a representation for T_C using $O(n')$ bits and supporting a large number of operations in constant time [16], in particular the preorder rank $r(v)$ of any node v. We also use structures that support two operations on bit-vectors and sequences X: $rank_a(X, i)$ is the number of occurrences of symbol a in $X[1..i]$, and $select_a(X, j)$ is the position of the jth occurrence of letter a in X.

We store a bit-vector $M[1..O(n')]$ in the same preorder of the nodes, where $M[r(v)] = \mathbf{1}$ iff node v is marked. Further, we store a string $S[1..O(n')]$ where we write down the level of each marked node, that is, $S[rank_1(M, r(v))] = \ell$ iff v is marked and of level ℓ. Operations *rank* and *select* on M can be supported in constant time and $o(|M|)$ further bits [6, 14]. Since there are $\lg^* n'$ distinct values of ℓ, the alphabet of S is small and S can be represented within $|S|H_0(S) + o(n')$ bits so that operations *rank* and *select* on S can be carried out in constant time [8]. Here $H_0(S)$ is the *zeroth-order empirical entropy* of S, defined as $|S|H_0(S) = \sum_\ell n_\ell \lg(|S|/n_\ell)$, where n_ℓ is the number of occurrences of symbol ℓ in S. Since $n_\ell \lg(|S|/n_\ell)$ is increasing[2] with n_ℓ and $n_\ell = O(n'/t_\ell^2)$ by Lemma 3, we have $|S|H_0(S) = O(n') \sum_\ell \lg(t_\ell^2)/t_\ell^2 = O(n') \sum_\ell \lg(t_\ell)/t_\ell^2 \le O(n') \sum_\ell 1/t_\ell = O(n')$.

With M and S we can create separate storage areas per level for the explicit P_v arrays of

[2] At least for $n_\ell \le |S|/e$. When n_ℓ is larger we can simply bound $n_\ell \lg(|S|/n_\ell) = O(n_\ell)$, thus we can remove all those large n_ℓ terms from the sum and add an extra $O(n')$ term to absorb them all.

marked nodes, each of which uses the same space for nodes of the same level: if a node v is marked (i.e., $M[r(v)] = \mathbf{1}$) and is of level $\ell = S[rank_1(M, r(v))]$, then we store its array P_v as the rth one in a separate sequence for level ℓ, where $r = rank_\ell(S, \ell)$.

Now consider unmarked nodes. The vectors o_v, r_v and b_v are concatenated in the same preorder of the nodes. While vectors o_v and r_v are of fixed size, vectors b_v are not. Their starting positions are thus indicated with $\mathbf{1}$s in a second bit-vector $B[1..O(n)]$. Given any original point $o_v[i] = \mathbf{1}$, it is the jth original point for $j = rank_1(o_v, i)$; recall that j is used to find d_j in b_v. Now b_v starts at position $select_1(B, r(v))$ in the concatenation of all the b_v's. Finally, we recover d_j as $select_0(b_v, j) - select_0(b_v, j-1)$.

Now we have to find the marked node v' leaving π at distance d_j from v. The strategy is to find the node u' that is "at the end" of π. More precisely, u' is a child of the lowest node of π and is the only node leaving π that is of the same level ℓ of v. Indeed, u' is the highest marked node of level ℓ in the subtree of v. Since we can compute node depth and level ancestors in constant time [16], we can compute the ancestor a of u' that is at depth $depth(v) + d_j - 1$, and find v' as the child of a that is not in π, that is, is not an ancestor of u'.

Now, to find u', we calculate the subtree size of v (in constant time [16]) and hence its level ℓ.[3] If the nodes are arranged in preorder, u' is the first node appearing after $r(v)$, $r(u') > r(v)$, which is marked $M[r(u')] = \mathbf{1}$ and whose level is $S[rank_1(M, r(u'))] = \ell$. This corresponds to the first occurrence of ℓ in S after position $rank_1(M, r(v))$. This is found in constant time with *rank* and *select* operations on S, and then $r(u')$ is found with *select* on M. Finally, the tree representation gives us u' from its rank $r(u')$ in constant time as well.

The sequence of colors c_π of path π is also associated with the last node u' of π, and all are concatenated in preorder of those nodes u'. As before, a bitmap is used to mark the starting position of each sequence c_π, and another bitmap is used to mark the preorders of the involved nodes u'.

Now let c_π be the sequence of $2|\pi|\kappa$ colors for path π, writing from highest to lowest node the 2κ colors of each node. The subarray corresponding to each v is easily found in c_π by knowing the depth of v and of u'. In order to find, given a position $c_\pi[i] = g'$, the largest $i' < i$ such that $c_\pi[i'] = g$, we build a monotone minimum perfect hash function (MMPHF) [1] for each original color g, recording the set of positions where either g or g' occur in c_π. A MMPHF can be regarded as a support for the limited operation $rank_{g,g'}(c_\pi, i)$ that counts the number of occurrences of g or g' in $c_\pi[1..i]$, provided $c_\pi[i] \in \{g, g'\}$. This is answered in constant time and using $O(|\pi|\kappa \lg\lg\kappa)$ bits. In addition, for each g we store a bit-vector c_π^g so that $c_\pi^g[rank_{g,g'}(c_\pi, i)] = \mathbf{1}$ iff $c_\pi[i] = g$. Then, after computing $r = rank_{g,g'}(c_\pi, i)$, we use *rank* and *select* on c_π^g to find the latest $\mathbf{1}$ in $c_\pi^g[1..r]$. This corresponds to the last occurrence of g preceding $c_\pi[i] = g'$. The position is mapped back from $c_\pi^g[o]$ to c_π using a sequence c'_π that identifies g' with g, so that the answer is $select_g(c'_\pi, o)$. We use a representation for c'_π that requires $O(|\pi|\kappa \lg\kappa)$ bits and gives constant *select* time [10]. Thus the structures representing paths π use space $O(|\pi|\kappa \lg\kappa)$, which is independent of the path level ℓ.

Extending access from cells to extents

We have shown how to provide constant-time access to the points in a cell. In order to extend this to the extent of a node v, we use the technique of [11] to find in constant time the 3 cells that form the extent of v, and simulate the concatenation of the 3 arrays P.

[3] To find the level in constant time from the subtree size, we can check directly for the case $\ell = 0$, and store the other answers in a small table of $\lg n'$ cells.

4 Predecessor Queries on P_v

Having constant-time access to P_v enables binary searching for the desired limits of the array A_v where the selection query is to be run. However the binary search time becomes the bottleneck. In this section we obtain fast predecessor searches that replace the binary search.

A classical predecessor structure uses $O(\kappa \lg n)$ bits, as the universe is the set of positions in A, and this adds up to $O(n \lg n)$ bits (note that this structure is needed in all the $O(n')$ nodes of T_C, not only the marked ones). A low-space predecessor structure when one has independent access to the sequence is the succinct SB-tree [12, Lem. 3.3]. For κ elements over a universe of size m, this structure supports predecessor queries in time $O(1 + \lg \kappa / \lg\lg m)$ using $O(\kappa \lg\lg m)$ bits, and a precomputed table of size $o(m)$ that depends only on m.

On a node v of level ℓ, the universe of positions is of size $O(\kappa\, s(v)) = O(\kappa\, t_{\ell-1}^2)$, thus the succinct SB-tree would use $O(\kappa \lg\lg(\kappa\, t_{\ell-1})) = O(\kappa \lg t_\ell + \kappa \lg\lg \kappa)$ bits. The first term is still too large, as just considering the nodes with $\ell = 1$ we add up to $O(n \lg\lg n)$ bits.

To improve on this, we will use a marking that is denser than that used in Section 3 (this marking is only used for the predecessor structures). We will further mark every $(t_\ell / \lg^2 t_\ell)$th node in the paths π of unmarked nodes of level ℓ. All marked nodes will store a succinct SB-tree. The number of marked nodes of level ℓ is now $O(n' \lg^2 t_\ell / t_\ell)$, so storing a succinct SB-tree in a each marked node of level ℓ adds up to $O(n \lg^3 t_\ell / t_\ell)$ bits. Adding up over all the levels ℓ we have $O(n) \sum_\ell \lg^3 t_\ell / t_\ell \leq O(n)(O(1) + \sum_{s \geq 0} s^3/2^s) = O(n)$ bits. The second term of the succinct SB-tree space, $O(\kappa \lg\lg \kappa)$, adds up to $O(n \lg\lg \kappa)$ bits.

As a result, the paths of unmarked nodes of level ℓ have length $O(t_\ell / \lg^2 t_\ell) = O(t_\ell)$. Consider one such path. The nodes leaving the path are of level $> \ell$, except the node u' leaving π at the bottom, which is of level ℓ. Therefore, we can divide the range of $s(v)$ split points covered by v into three areas: (1) the area covered by the subtrees that leave π to the left, (2) the area covered by the subtrees that leave π to the right, and (3) the area covered by u'. Each of those areas is contiguous, (1) preceding (3) preceding (2). Since there are $O(t_\ell)$ nodes of type (1) and each is of level at least $\ell+1$, the total area covered by those is of size $O(t_\ell \cdot \kappa\, t_\ell^2) = O(\kappa\, t_\ell^3)$. The case of (2) is analogous. Therefore, for the (unmarked) nodes on π we store a succinct SB-tree for the values in area (1) and another for the values in area (2), both using $O(\kappa \lg\lg(\kappa\, t_\ell^3)) = O(\kappa \lg\lg(\kappa\, t_\ell))$ bits. Given a predecessor request, we first find the node u' below π as in Section 3, and determine in constant time whether the query falls in the area (1), (2), or (3) (by obtaining the limits $[x_l + 1, x_r]$ of u', as explained). If it falls in areas (1) or (2) we use the corresponding succinct SB-tree of v, otherwise we use the succinct SB-tree of u' (which is marked and hence stores a regular succinct SB-tree). We use the same techniques as in Section 3 to store and access the (variable-sized) representations of the succinct SB-trees.

With this twist, the space over a node of level ℓ is $O(\kappa \lg\lg(\kappa\, t_\ell))$ bits, adding up to at most $O(n \lg\lg\lg n + n \lg\lg \kappa)$ bits, again dominated by the nodes of level $\ell = 1$. This gives a total space of $O(n(\lg \kappa + \lg\lg\lg n))$ and a time of $O(\lg \kappa / \lg\lg n)$. Note that the time is improved from $O(\lg \kappa / \lg\lg t_\ell)$ to $O(\lg \kappa / \lg\lg n)$ by using the same precomputed table over a universe of size n for all the nodes, and this table requires $o(n)$ further bits. This result is already as desired if $\lg \kappa = \Omega(\lg\lg\lg n)$. In the sequel we address the case $\kappa = O(\lg\lg n)$.

4.1 Handling Small κ Values

When $\kappa = O(\lg\lg n)$ we will not use the mechanism of storing succinct SB-trees for areas (1) and (2) of unmarked nodes as before, but a different mechanism. Let π be a path of unmarked nodes of level ℓ. Let $u_1, u_2, \ldots$ be the nodes that leave π from the left, reading their areas in left-to-right order (i.e., top-down in π), and $v_1, v_2, \ldots$ be the nodes that leave π from the right, also reading them in left-to-right order (i.e., bottom-up in π). Then the area of A covered by π can be partitioned into the $|\pi|$ consecutive areas covered by $u_1, u_2, \ldots, u', v_1, v_2, \ldots$. All those nodes are marked and thus store their own succinct SB-tree.

Our problem is to determine, given a node v in π, which is the predecessor in P_v of a given position p. A first predecessor structure, associated with π, determines in which of those $|\pi|$ areas p belongs (the node containing that area will descend from v). Let ℓ_i be the level of node u_i. Then the area covered by u_i is of length $O(\kappa\, t_{\ell_i - 1}^2)$. Thus we can encode those lengths with, say, γ-codes [2], within $O(\sum_i \lg(\kappa\, t_{\ell_i-1}^2)) = O(|\pi| \lg \kappa + \sum_i t_{\ell_i})$ bits.

From a space accounting point of view, this space can be afforded because we can charge $O(\lg \kappa + t_{\ell_i})$ bits to the storage of u_i. As u_i's level is larger than p, it is a marked node (see Section 3). Thus there are $O(n'/t_{\ell_i}^2)$ such nodes overall, each of which will be charged $O(t_{\ell_i})$ bits only once, from the path π it leaves, for a total of $O(n'/t_{\ell_i})$ bits, adding up to $O(n')$ bits overall. For the other term, note that we can always afford $\lg \kappa$ bits of space per node.

On the other hand, we note that, since $\ell_i > \ell$, it holds $O(|\pi| \lg \kappa + \sum_i t_{\ell_i}) = O(|\pi| \lg \kappa + |\pi| \lg t_\ell)$. Since $|\pi| = O(t_\ell / \lg^2 t_\ell)$, $t_\ell = O(\lg n)$ even for $\ell = 1$, and $\kappa = O(\lg \lg n)$, the space is $O(\lg n / \lg \lg n) = o(\lg n)$, and thus the whole description of the u_i areas fits in a single computer word, and a global precomputed table of $o(n)$ bits can be used to answer any predecessor query in constant time.

We proceed analogously with the areas of $v_1, v_2, \ldots$. Now, a predecessor query for the areas $u_1, u_2, \ldots, u', v_1, v_2, \ldots$ can be answered as before: We first determine whether the answer is u' with a constant number of comparisons, and if not, we use the global precomputed table with the description of the lengths of the areas of the u_i or the v_i nodes. This takes $O(1)$ time. Once we know the area where the answer lies, we use the succinct SB-tree of the corresponding node v' (which we remind it is marked) to find the position of the predecessor in its $P_{v'}$ array. Node v' is found by first computing its parent v'' with level ancestor queries from u' (found as in Section 3) and then v' is the child of v'' not in π.

Once we have that the predecessor of p in v' is $P_{v'}[o']$, the final challenge is to map that position in v' to the corresponding position in v. We will reuse the encoding of 4κ colors described in Section 3. Note that, in the string of $2|\pi|\kappa$ colors associated with the path π, we have sufficient information to determine which of the points in v are inherited in v': if the color of the point is g or g', we track g' downwards in π until it does not appear in some node v'', then the point is inherited in the sibling v' of v'' not in π. Note that all the points of v that are inherited in v' are contiguous in P_v.

In addition to the color information c_v, we store associated with v a sequence of numbers $n_v[1..2\kappa]$, so that $n_v[i]$ is the rank of the ith point of v among the points stored in v', where v' is the first node leaving π that inherits the ith point of v. With the information of c_v and n_v, and given the predecessor of a point in $P_{v'}$, we have sufficient information to determine the predecessor of the point in P_v: only some of the points of $P_{v'}$ are inherited from P_v.

The set of all c_v and n_v arrays in π add up to $O(|\pi|\kappa \lg \kappa)$ bits, and since $|\pi| = O(t_\ell / \lg^2 t_\ell)$, $t_\ell = O(\lg n)$, and $\kappa = O(\lg \lg n)$, this is $O(\lg n \lg \lg \lg n / \lg \lg n) = o(\lg n)$. Thus a global precomputed table of $o(n)$ bits can precompute all the process of determining the predecessor in any v given that the answer is at any position in any descendant v'.

Predecessors on extents

Once again, P_v refers to the extent of v, not only to its cell, whereas we support predecessors only on the points of the cell. With a couple of comparisons we determine whether the predecessor query must be run on the cell of v or on the cell of a neighboring node.

5 Wrapping Up

We can now describe a structure that, given a value κ, uses $O(n \lg \kappa)$ bits and answers a query $\mathsf{select}(i, j, k)$ for any $k \le \kappa$ in time $O(1 + \lg \kappa / \lg \lg n)$, as follows:

1. We find the maximal interval $[l, r]$ such that $i \le x_l + 1 \le x_r \le j$, using *rank*/*select* on a bit-vector that marks the split points x_s [11].
2. If the interval is empty, then $A[i..j]$ is contained in a leaf of T_C, which covers $O(\kappa)$ consecutive values of A. Then the query can be directly run on plain range selection structures [4] associated with each leaf (these structures add up to $O(n \lg \kappa)$ bits).
3. Otherwise, we find the highest node $v \in T_C$ containing $[x_l + 1, x_r]$, as well as the other two neighbor nodes that span the extent of v, all in constant time [11].
4. Using the structures of Section 4, we find the predecessor $P_v[r]$ of j, and the successor $P_v[l]$ of i (the successor needs structures analogous to the predecessor), in time $O(1 + \lg \kappa / \lg \lg n)$.
5. We use the range selection structure [4] associated with P_v to run the query $o = \mathsf{select}(l, r, k)$. The time is $O(1 + \lg_w \kappa)$.
6. We use the structures of Section 3 to compute the final answer $P_v[o]$, in $O(1)$ time, adding to it the starting offset of node v.

In order to reduce the time to $O(1 + \lg k / \lg \lg n)$, we build our data structures for values $\kappa_t = 2^{2^t}$, for $t = 0, 1, \ldots, \tau$, where τ is such that $2^{2^{\tau-1}} < \kappa \le 2^{2^\tau}$. The space for those structures is $O(n) \sum_{t=0}^{\tau} \lg \kappa_t = O(n) \sum_{t=0}^{\tau} 2^t = O(n\, 2^\tau) = O(n \lg \kappa)$. A query $\mathsf{select}(i, j, k)$ is run on the structure for κ_t such that $\kappa_{t-1} < k \le \kappa_t$, that is, $2^{t-1} < \lg k \le 2^t$,[4] and thus its query time is $O(1 + \lg \kappa_t / \lg \lg n) = O(1 + 2^t / \lg \lg n) = O(1 + \lg k / \lg \lg n)$. This proves Theorem 1.

Answering the query $\mathsf{top}(i, j, k)$

We proceed as for query $\mathsf{select}(i, j, k)$ until we find the kth largest element in $A_v[l..r]$, let it be $A_v[o]$. Now we must find all the elements $A_v[s]$ in $A_v[l..r]$ where $A_v[s] \ge A_v[o]$. With an RMQ structure over A_v we can do this using Muthukrishnan's algorithm [15]: find the maximum in $A_v[l..r]$, let it be $A_v[m_1]$, then continue recursively with $A_v[l..m_1 - 1]$ and $A_v[m_1 + 1..r]$ stoping the recursion when the maximum found at $A_v[m]$ satisfies $A_v[m] < A_v[o]$. Recall that A_v is a permutation on $O(\kappa)$ symbols and thus we can afford storing it directly. Finally, when we have the positions $m_1, \ldots, m_k$ of the top-k elements, we return $P_v[m_1], \ldots, P_v[m_k]$. The overall time is $O(\lg k / \lg \lg n + k) = O(k)$. This proves Theorem 2.

Note that we deliver the top-k elements in unsorted order. On the other hand, after $O(1 + \lg k / \lg \lg n)$ time, each new result is delivered in $O(1)$ time.

6 Conclusions

We have shown how to build an encoding data structure that uses asymptotically optimal space of $O(n \lg \kappa)$ bits that answers κ-bounded rank range selection queries in time $O(1 + \lg k / \lg \lg n)$, and range top-k queries in $O(k)$ time for any $k \le \kappa$. It would be interesting to obtain exactly optimal space (to within lower-order terms), but the precise lower bound is unknown even for $k = 2$ [7]. It would also be interesting to obtain optimal time bounds for the general case $w = \Omega(\lg n)$.

[4] The search for the right t can be done in constant time by computing $\lg \lg k$ and consulting a small precomputed table of $\lg \lg K \le \lg \lg n$ entries.

References

1 D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna. Monotone minimal perfect hashing: searching a sorted table with $o(1)$ accesses. In *Proc. 20th SODA*, pages 785–794, 2009.

2 T. Bell, J. Cleary, and I. Witten. *Text compression.* Prentice Hall, 1990.

3 G. S. Brodal, B. Gfeller, A. G. Jørgensen, and P. Sanders. Towards optimal range medians. *Theor. Comp. Sci.*, 412(24):2588–2601, 2011.

4 G. S. Brodal and A.G. Jørgensen. Data structures for range median queries. In *Proc. 20th ISAAC*, LNCS 5878, pages 822–831, 2009.

5 T. Chan and B.T. Wilkinson. Adaptive and approximate orthogonal range counting. In *Proc. 24th SODA*, pages 241–251, 2013.

6 D. Clark. *Compact Pat Trees.* PhD thesis, University of Waterloo, Canada, 1996.

7 P. Davoodi, G. Navarro, R. Raman, and S. Srinivasa Rao. Encoding range minima and range top-2 queries. *Philosphical Transactions of the Royal Society A*, 372:20130131, 2014.

8 P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Alg.*, 3(2):article 20, 2007.

9 J. Fischer and V. Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comp.*, 40(2):465–492, 2011.

10 A. Golynski, I. Munro, and S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proc. 17th SODA*, pages 368–373, 2006.

11 R. Grossi, J. Iacono, G. Navarro, R. Raman, and S. Srinivasa Rao. Encodings for range selection and top-k queries. In *Proc. 21st ESA*, LNCS 8125, pages 553–564, 2013.

12 R. Grossi, A. Orlandi, R. Raman, and S. Srinivasa Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *Proc. 26th STACS*, pages 517–528, 2009.

13 A. G. Jørgensen and K. G. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. 22nd SODA*, pages 805–813, 2011.

14 I. Munro. Tables. In *Proc. 16th FSTTCS*, LNCS 1180, pages 37–42, 1996.

15 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc 13th SODA*, pages 657–666, 2002.

16 K. Sadakane and G. Navarro. Fully-functional succinct trees. In *Proc. 21st SODA*, pages 134–149, 2010.

17 B. T. Wilkinson. Personal Communication, 2014.

Output-Sensitive Pattern Extraction in Sequences

Roberto Grossi[1], Giulia Menconi[1], Nadia Pisanti[1], Roberto Trani[1], and Søren Vind*[2]

1 **Università di Pisa, Dipartimento di Informatica**
`grossi@di.unipi.it, menconigiulia@gmail.com, pisanti@di.unipi.it, tranir@cli.di.unipi.it`

2 **Technical University of Denmark, DTU Compute**
`sovi@dtu.dk`

Abstract

Genomic Analysis, Plagiarism Detection, Data Mining, Intrusion Detection, Spam Fighting and Time Series Analysis are just some examples of applications where extraction of recurring patterns in sequences of objects is one of the main computational challenges. Several notions of patterns exist, and many share the common idea of strictly specifying some parts of the pattern and to *don't care* about the remaining parts. Since the number of patterns can be exponential in the length of the sequences, *pattern extraction* focuses on statistically relevant patterns, where any attempt to further refine or extend them causes a loss of significant information (where the number of occurrences changes). Output-sensitive algorithms have been proposed to enumerate and list these patterns, taking polynomial time $O(n^c)$ per pattern for constant $c > 1$, which is impractical for massive sequences of very large length n.

We address the problem of extracting maximal patterns with at most k don't care symbols and at least q occurrences. Our contribution is to give the first algorithm that attains a *stronger* notion of output-sensitivity, borrowed from the analysis of data structures: the cost is proportional to the *actual* number of occurrences of each pattern, which is at most n and practically much smaller than n in real applications, thus avoiding the aforementioned cost of $O(n^c)$ per pattern.

1998 ACM Subject Classification E.1 Data Structures

Keywords and phrases Pattern Extraction, Motif Detection, Pattern Discovery, Motif Trie

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.303

1 Introduction

In *pattern extraction*, the task is to extract the "most important" and frequently occurring patterns from sequences of "objects" such as log files, time series, text documents, datasets or DNA sequences. Each individual object can be as simple as a character from {`A`, `C`, `G`, `T`} or as complex as a `json` record from a log file. What is of interest to us is the potentially very large set of all possible different objects, which we call the *alphabet* Σ, and sequence S built with n objects drawn from Σ.

We define the occurrence of a pattern in S as in *pattern matching* but its importance depends on its statistical relevance, namely, if the number of occurrences is above a certain threshold. However, pattern extraction is not to be confused with pattern matching. The problems may be considered inverse of each other: the former gets an input sequence S from the user, and extracts patterns P and their occurrences from S, where both are unknown

* Supported by a grant from the Danish National Advanced Technology Foundation.

to the user; the latter gets S and a given pattern P from the user, and searches for P's occurrences in S, and thus only the pattern occurrences are unknown to the user.

Many notions of patterns exist, reflecting the diverse applications of the problem [11, 4, 19, 21]. We study a natural variation allowing the special don't care character $\star$ in a pattern to mean that the position inside the pattern occurrences in S can be ignored (so $\star$ matches any single character in S). For example, $\texttt{TA} \star \texttt{C} \star \texttt{ACA} \star \texttt{GTG}$ is a pattern for DNA sequences.

A *motif* is a pattern of *any* length with *at most k don't cares* occurring *at least q times* in S. In this paper, we consider the problem of determining the *maximal* motifs, where any attempt to extend them or replace their $\star$'s with symbols from Σ causes a loss of significant information (where the number of occurrences in S changes). We denote the family of all motifs by M_{qk}, the set of maximal motifs $\mathcal{M} \subseteq M_{qk}$ (dropping the subscripts in $\mathcal{M}$) and let $\text{occ}(m)$ denote the number of occurrences of a motif m inside S. It is well known that M_{qk} can be exponentially larger than $\mathcal{M}$ [16].

Our Results. We show how to efficiently build an index that we call a *motif trie* which is a trie that contains all prefixes, suffixes and occurrences of $\mathcal{M}$, and we show how to extract $\mathcal{M}$ from it. The motif trie is built in level-wise, using an oracle $\textsc{Generate}(u)$ that reveals the children of a node u efficiently using properties of the motif alphabet and a bijection between new children of u and intervals in the ordered sequence of occurrences of u. We are able to bound the resulting running time with a strong notion of *output-sensitive* cost, borrowed from the analysis of data structures, where the cost is proportional to the *actual* number $\text{occ}(m)$ of occurrences of each maximal motif m.

▶ **Theorem 1.** *Given a sequence S of n objects over an alphabet Σ, and two integers $q > 1$ and $k \geq 0$, there is an algorithm for extracting the maximal motifs $\mathcal{M} \subseteq M_{qk}$ and their occurrences from S in $O\Big(n(k + \log \Sigma) + (k+1)^3 \times \sum_{m \in \mathcal{M}} \text{occ}(m)\Big)$ time.*

Our result may be interesting for several reasons. First, observe that this is an optimal listing bound when the maximal number of don't cares is $k = O(1)$, which is true in many practical applications. The resulting bound is $O(n \log \Sigma + \sum_{m \in \mathcal{M}} \text{occ}(m))$ time, where the first additive term accounts for building the motif trie and the second term for discovering and reporting all the occurrences of each maximal motif.

Second, our bound provides a strong notion of output-sensitivity since it depends on how many times each maximal motif occurs in S. In the literature for enumeration, an output-sensitive cost traditionally means that there is polynomial cost of $O(n^c)$ per pattern, for a constant $c > 1$. This is infeasible in the context of big data, as n can be very large, whereas our cost of $\text{occ}(m) \leq n$ compares favorably with $O(n^c)$ per motif m, and $\text{occ}(m)$ can be actually much smaller than n in practice. This has also implications in what we call "the `CTRL-C` argument," which ensures that we can safely stop the computation for a *specific* sequence S if it is taking too much time[1]. Indeed, if much time is spent with our solution, too many results to be really useful may have been produced. Thus, one may stop the computation and refine the query (change q and k) to get better results. On the contrary, a non-output-sensitive algorithm may use long time without producing any output: It does not indicate if it may be beneficial to interrupt and modify the query.

Third, our analysis improves significantly over the brute-force bound: M_{qk} contains pattern candidates of lengths p from 1 to n with up to $\min\{k, p\}$ don't cares, and so has size

[1] Such an algorithm is also called an anytime algorithm in the literature.

$\sum_p |\Sigma|^p \times (\sum_{i=1}^{\min\{k,p\}} \binom{p}{i}) = O(|\Sigma|^n n^k)$. Each candidate can be checked in $O(nk)$ time (e. g. string matching with k mismatches), or $O(k)$ time if using a data structure such as the suffix tree [19]. In our analysis we are able to remove both of the nasty exponential dependencies on $|\Sigma|$ and n in $O(|\Sigma|^n n^k)$. In the current scenario where implementations are fast in practice but skip worst-case analysis, or state the latter in pessimistic fashion equivalent to the brute-force bound, our analysis could explain why several previous algorithms are fast in practice. (We have implemented a variation of our algorithm that is very fast in practice.)

Related Work. Although the literature on pattern extraction is vast and spans many different fields of applications with various notation, terminology and variations, we could not find time bounds explicitly stated obeying our stronger notion of output-sensitivity, even for pattern classes different from ours. Output-sensitive solutions with a polynomial cost per pattern have been previously devised for slightly different notions of patterns. For example, Parida et al. [15] describe an enumeration algorithm with $O(n^2)$ time per maximal motif plus a bootstrap cost of $O(n^5 \log n)$ time. [2] Arimura and Uno obtain a solution with $O(n^3)$ delay per maximal motif where there is no limitations on the number of don't cares [4]. Similarly, the MadMX algorithm [11] reports dense motifs, where the ratio of don't cares and normal characters must exceed some threshold, in time $O(n^3)$ per maximal dense motif. Our stronger notion of output-sensitivity is borrowed from the design and analysis of data structures, where it is widely employed. For example, searching a pattern P in S using the suffix tree [14] has cost proportional to P's length and its number of occurrences. A one-dimensional query in a sorted array reports all the wanted keys belonging to a range in time proportional to their number plus a logarithmic cost. Therefore it seemed natural to us to extend this notion to enumeration algorithms also.

Applications. Although the pattern extraction problem has found immediate applications in stringology and biological sequences, it is highly multidisciplinary and spans a vast number of applications in different areas. This situation is similar to the one for the edit distance problem and dynamic programming. We here give a short survey of some significant applications, but others are no doubt left out due to the difference in terminology used (see [1] for further references). In computational biology, motif discovery in biological sequences identifies areas of interest[19, 21, 11, 1]. Computer security researches use patterns in log files to perform intrusion detection and find attack signatures based on their frequencies [9], while commercial anti-spam filtering systems use pattern extraction to detect and block SPAM [18]. In the data mining community pattern extraction is used extensively [13] as a core method in web page content extraction [7] and time series analysis [17, 20]. In plagiarism detection finding recurring patterns across a (large) number of documents is a core primitive to detect if significant parts of documents are plagiarized [6] or duplicated [5, 8]. And finally, in data compression extraction of the common patterns enables a compression scheme that competes in efficiency with well-established compression schemes [3].

As the motif trie is an index, we believe that it may be of independent interest for storing similar patterns across similar strings. Our result easily extends to real-life applications requiring a solution with two thresholds for motifs, namely, on the number of occurrences in a sequence and across a minimum number of sequences.

[2] The set intersection problem (SIP) in appendix A of [15] requires polynomial time $O(n^2)$: The recursion tree of depth $\leq n$ can have unary nodes, and each recursive call requires $O(n)$ to check if the current subset has been already generated.

String	`TACTGACACTGCCGA`
Quorum	$q = 2$
Don't cares	$k = 1$

(a) Input and parameters for example.

Maximal Motif	Occurrence List
`A`	2, 6, 8, 15
`AC`	2, 6, 8
`ACTG⋆C`	2, 8
`C`	3, 7, 9, 12, 13
`G`	5, 11, 14
`GA`	5, 14
`G⋆C`	5, 11
`T`	1, 4, 10
`T⋆C`	1, 10

(b) Output: Maximal motifs found (and their occurrence list) for the given input.

Figure 1 Example 1: Maximal Motifs found in string.

Reading Guide. Our solution has two natural parts. In Section 3 we define the *motif trie*, which is an index storing all maximal motifs and their prefixes, suffixes and occurrences. We show how to report only the maximal motifs in time linear in the size of the trie. That is, it is easy to extract the maximal motifs from the motif trie – the difficulty is to build the motif trie without knowing the motifs in advance. In Section 4 and 5 we give an efficient algorithm for constructing the motif trie and bound its construction time by the number of occurrences of the maximal motifs, thereby obtaining an output-sensitive algorithm.

2 Preliminaries

Strings. We let Σ be the alphabet of the input string $S \in \Sigma^*$ and $n = |S|$ be its length. For $1 \leq i \leq j \leq n$, $S[i,j]$ is the substring of S between index i and j, both included. $S[i,j]$ is the empty string ε if $i > j$, and $S[i] = S[i,i]$ is a single character. Letting $1 \leq i \leq n$, a prefix or suffix of S is $S[1,i]$ or $S[i,n]$, respectively. The *longest common prefix* $lcp(x,y)$ is the longest string such that $x[1,|\,lcp(x,y)|] = y[1,|\,lcp(x,y)|]$ for any two strings $x, y \in \Sigma^*$.

Tries. A trie T over an alphabet Π is a rooted, labeled tree, where each edge (u,v) is labeled with a symbol from Π. All edges to children of node $u \in T$ must be labeled with distinct symbols from Π. We may consider node $u \in T$ as a string generated over Π by spelling out characters from the root on the path towards u. We will use u to refer to both the node and the string it encodes, and $|u|$ to denote its string length. A property of the trie T is that for any string $u \in T$, it also stores all prefixes of u. A compacted trie is obtained by compacting chains of unary nodes in a trie, so the edges are labeled with substrings: the suffix tree for a string is special compacted trie that is built on all suffixes of the string [14].

Motifs. A motif $m \in \Sigma\,(\Sigma \cup \{\star\})^*\,\Sigma$ consist of symbols from Σ and *don't care characters* $\star \notin \Sigma$. We let the length $|m|$ denote the number of symbols from $\Sigma \cup \{\star\}$ in m, and let $\mathrm{dc}(m)$ denote the number of $\star$ characters in m. Motif m *occurs* at position p in S iff $m[i] = S[p+i-1]$ or $m[i] = \star$ for all $1 \leq i \leq |m|$. The number of occurrences of m in S is denoted $\mathrm{occ}(m)$. Note that appending $\star$ to either end of a motif m does not change $\mathrm{occ}(m)$, so we assume that motifs starts and ends with symbols from Σ. A *solid block* is a maximal (possibly empty ε) substring from Σ^* inside m.

We say that a motif m can be *extended* by adding don't cares and characters from Σ to either end of m. Similarly, a motif m can be *specialized* by replacing a don't care $\star$ in m with a symbol $c \in \Sigma$. An example is shown in Figure 1.

Maximal Motifs. Given an integer quorum $q > 1$ and a maximum number of don't cares $k \geq 0$, we define a family of motifs M_{qk} containing motifs m that have a limited number of don't cares $\mathrm{dc}(m) \leq k$, and occurs frequently $\mathrm{occ}(m) \geq q$. A *maximal motif* $m \in M_{qk}$ cannot be extended or specialized into another motif $m' \in M_{qk}$ such that $\mathrm{occ}(m') = \mathrm{occ}(m)$. Note that extending a maximal motif m into motif $m'' \notin M_{qk}$ may maintain the occurrences (but have more than k don't cares). We let $\mathcal{M} \subseteq M_{qk}$ denote the *set of maximal motifs*.

Motifs $m \in M_{qk}$ that are *left-maximal* or *right-maximal* cannot be specialized or extended on the left or right without decreasing the number of occurrences, respectively. They may, however, be prefix or suffix of another (possibly maximal) $m' \in M_{qk}$, respectively.

▶ **Fact 1.** *If motif $m \in M_{qk}$ is right-maximal then it is a suffix of a maximal motif.*

3 Motif Tries and Pattern Extraction

This section introduces the *motif trie*. This trie is not used for searching but its properties are exploited to orchestrate the search for maximal motifs in $\mathcal{M}$ to obtain a strong output-sensitive cost. Due to space constraints, all proofs have been omitted in the present version.

3.1 Efficient Representation of Motifs

We first give a few simple observations that are key to our algorithms. Consider a suffix tree built on S over the alphabet Σ, which can be done in $O(n \log |\Sigma|)$ time. It is shown in [21, 10] that when a motif m is maximal, its solid blocks correspond to nodes in the suffix tree for S, matching their substrings from the root[3]. For this reason, we introduce a new alphabet, the *solid block alphabet* Π of size at most $2n$, consisting of the strings stored in all the suffix tree nodes.

We can write a maximal motif $m \in M_{qk}$ as an alternating sequence of $\leq k+1$ solid blocks and $\leq k$ don't cares, where the first and last solid block must be different from ε. Thus we represent m as a sequence of $\leq k+1$ strings from Π since the don't cares are implicit. By traversing the suffix tree nodes in *preorder* we assign integers to the strings in Π, allowing us to assume that $\Pi \subseteq [1, \ldots, 2n]$, and so each motif $m \in M_{qk}$ is actually represented as a sequence of $\leq k+1$ integers from 1 to $|\Pi| = O(n)$. Note that the order on the integers in Π shares the following grouping property with the strings over Σ.

▶ **Lemma 2.** *Let A be an array storing the sorted alphabet Π. For any string $x \in \Sigma^*$, the solid blocks represented in Π and sharing x as a common prefix, if any, are grouped together in A in a contiguous segment $A[i, j]$ for some $1 \leq i \leq j \leq |\Pi|$.*

When it is clear from its context, we will use the shorthand $x \in \Pi$ to mean equivalently a string x represented in Π or the integer x in Π that represents a string stored in a suffix tree node. We observe that the set of strings represented in Π is *closed* under the longest common prefix operation: for any $x, y \in \Pi$, $lcp(x, y) \in \Pi$ and it may be computed in constant time after augmenting the suffix tree for S with a lowest common ancestor data structure [12].

Summing up, the above relabeling from Σ to Π only requires the string $S \in \Sigma^*$ and its suffix tree augmented with lowest common ancestor information.

[3] The proofs in [21, 10] can be easily extended to our notion of maximality.

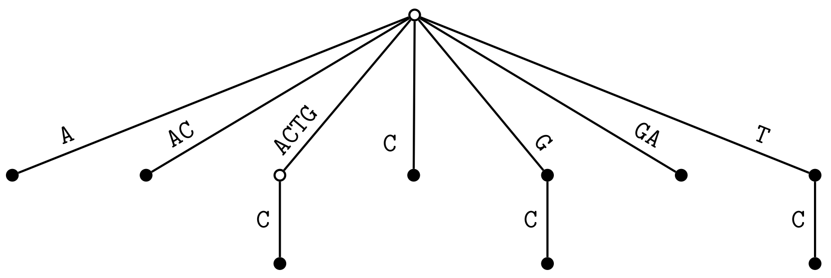


Figure 2 Motif trie for Example 1. The black nodes are maximal motifs (with their occurrence lists shown in Fig. 1(b)).

3.2 Motif Tries

We now exploit the machinery on alphabets described in Section 3.1. For the input sequence S, consider the family M_{qk} defined in Section 2, where each m is seen as a string $m = m[1, \ell]$ of $\ell \leq k+1$ integers from 1 to $|\Pi|$. Although each m can contain $O(n)$ symbols from Σ, we get a benefit from treating m as a short string over Π: unless specified otherwise, the prefixes and suffixes of m are respectively $m[1, i]$ and $m[i, \ell]$ for $1 \leq i \leq \ell$, where $\ell = \mathrm{dc}(m) + 1 \leq k + 1$. This helps with the following definition as it does not depend on the $O(n)$ symbols from Σ in a maximal motif m but it solely depends on its $\leq k+1$ length over Π.

▶ **Definition 3** (Motif Trie). A *motif trie* T is a trie over alphabet Π which stores all maximal motifs $\mathcal{M} \subseteq M_{qk}$ and their suffixes.

As a consequence of being a trie, T implicitly stores all prefixes of all the maximal motifs and edges in T are labeled using characters from Π. Hence, all sub-motifs of the maximal motifs are stored in T, and the motif trie can be essentially seen as a generalized suffix trie[4] storing $\mathcal{M}$ over the alphabet Π. From the definition, T has $O((k+1) \cdot |\mathcal{M}|)$ leaves, the total number of nodes is $O(|T|) = O((k+1)^2 \cdot |\mathcal{M}|)$, and the height is at most $k+1$.

We may consider a node u in T as a string generated over Π by spelling out the $\leq k+1$ integers from the root on the path towards u. To decode the motif stored in u, we retrieve these integers in Π and, using the suffix tree of S, we obtain the corresponding solid blocks over Σ and insert a don't care symbol between every pair of consecutive solid blocks. When it is clear from the context, we will use u to refer to *(1)* the node u or *(2)* the string of integers from Π stored in u, or *(3)* the corresponding motif from $(\Sigma \cup \{\star\})^*$. We reserve the notation $|u|$ to denote the length of motif u as the number of characters from $\Sigma \cup \{\star\}$. Each node $u \in T$ stores a list L_u of occurrences of motif u in S, i. e. u occurs at p in S for $p \in L_u$.

Since child edges for $u \in T$ are labeled with solid blocks, the child edge labels may be prefixes of each other, and one of the labels may be the empty string ε (which corresponds to having two neighboring don't cares in the decoded motif).

3.3 Reporting Maximal Motifs using Motif Tries

Suppose we are given a motif trie T but we do not know which nodes of T store the maximal motifs in S. We can identify and report the maximal motifs in T in $O(|T|) = O((k+1)^2 \cdot |\mathcal{M}|) = O((k+1)^2 \cdot \sum_{m \in \mathcal{M}} \mathrm{occ}(m))$ time as follows.

[4] As it will be clear later, a compacted motif trie does not give any advantage in terms of the output-sensitive bound compared to the motif trie.

We first identify the set R of nodes $u \in T$ that are right-maximal motifs. A characterization of right-maximal motifs in T is relatively simple: we choose a node $u \in T$ if *(i)* its parent edge label is not ε, and *(ii)* u has no descendant v with a non-empty parent edge label such that $|L_u| = |L_v|$. By performing a bottom-up traversal of nodes in T, computing for each node the length of the longest list of occurrences for a node in its subtree with a non-empty edge label, it is easy to find R in time $O(|T|)$ and by Fact 1, $|R| = O((k+1) \cdot |\mathcal{M}|)$.

Next we perform a radix sort on the set of pairs $\langle |L_u|, \text{reverse}(u)\rangle$, where $u \in R$ and reverse(u) denotes the reverse of the string u, to select the motifs that are also left-maximal (and thus are maximal). In this way, the suffixes of the maximal motifs become prefixes of the reversed maximal motifs. By Lemma 2, those motifs sharing common prefixes are grouped together consecutively. However, there is a caveat, as one maximal motif m' could be a suffix of another maximal motif m and we do not want to drop m': in that case, we have that $|L_m| \neq |L_{m'}|$ by the definition of maximality. Hence, after sorting, we consider consecutive pairs $\langle |L_{u_1}|, \text{reverse}(u_1)\rangle$ and $\langle |L_{u_2}|, \text{reverse}(u_2)\rangle$ in the order, and eliminate u_1 iff $|L_{u_1}| = |L_{u_2}|$ and u_1 is a suffix of u_2 in time $O(k+1)$ per pair (i. e. prefix under reverse). The remaining motifs are maximal.

4 Building Motif Tries

The goal of this section is to show how to efficiently build the motif trie T discussed in Section 3.2. Suppose without loss of generality that enough new symbols are prepended and appended to the sequence S to avoid border cases. We want to store the maximal motifs of S in T as strings of length $\leq k+1$ over Π. Some difficulties arise as we do not know in advance which are the maximal motifs. Actually, we plan to find them *during* the output-sensitive construction of T, which means that we would like to obtain a construction bound close to the term $\sum_{m \in \mathcal{M}} \text{occ}(m)$ stated in Theorem 1.

We proceed in top-down and level-wise fashion by employing an *oracle* that is invoked on each node u on the last level of the partially built trie, and which reveals the future children of u. The oracle is executed many times to generate T level-wise starting from its root u with $L_u = \{1, \ldots, n\}$, and stopping at level $k+1$ or earlier for each root-to-node path. Interestingly, this sounds like the wrong way to do anything efficiently, e. g. it is a slow way to build a suffix tree, however the oracle allows us to amortize the total cost to construct the trie. In particular, we can bound the total cost by the total number of occurrences of the maximal motifs stored in the motif trie.

The oracle is implemented by the GENERATE(u) procedure that generates the children $u_1, \ldots, u_d$ of u. We ensure that *(i)* GENERATE(u) operates on the $\leq k+1$ length motifs from Π, and *(ii)* GENERATE(u) avoids generating the motifs in $M_{qk} \setminus \mathcal{M}$ that are not suffixes or prefixes of maximal motifs. This is crucial, as otherwise we cannot guarantee output-sensitive bounds because M_{qk} can be exponentially larger than $\mathcal{M}$.

In Section 5 we will show how to implement GENERATE(u) and prove:

▶ **Lemma 4.** *Algorithm* GENERATE(u) *produces the children of* u *and can be implemented in time* $O(\text{sort}(L_u) + (k+1) \cdot |L_u| + \sum_{i=1}^{d} |L_{u_i}|)$.

By summing the cost to execute procedure GENERATE(u) for all nodes $u \in T$, we now bound the construction time of T. Observe that when summing over T the formula stated in Lemma 4, each node exists once in the first two terms and once in the third term, so the latter can be ignored when summing over T (as it is dominated by the other terms)

$$\sum_{u\in T}(\mathrm{sort}(L_u) + (k+1)\cdot|L_u| + \sum_{i=1}^{d}|L_{u_i}|) = O\left(\sum_{u\in T}(\mathrm{sort}(L_u) + (k+1)\cdot|L_u|)\right) \quad .$$

We bound

$$\sum_{u\in T}\mathrm{sort}(L_u) = O\left(n(k+1) + \sum_{u\in T}|L_u|\right)$$

by running a single cumulative radix sort for all the instances over the several nodes u at the same level, allowing us to amortize the additive cost $O(n)$ of the radix sorting among nodes at the same level (and there are at most $k+1$ such levels).

To bound $\sum_{u\in T}|L_u|$, we observe $\sum_i |L_{u_i}| \geq |L_u|$ (as trivially the ε extension always maintains the number of occurrences of its parent). Consequently we can charge each leaf u the cost of its $\leq k$ ancestors, so

$$\sum_{u\in T}|L_u| = O\left((k+1)\times\sum_{\text{leaf } u\in T}|L_u|\right) \quad .$$

Finally, from Section 3.2 there cannot be more leaves than maximal motifs in $\mathcal{M}$ and their suffixes, and the occurrence lists of maximal motifs dominate the size of the non-maximal ones in T, which allows us to bound:

$$(k+1)\times\sum_{\text{leaf } u\in T}|L_u| = O\left((k+1)^2\times\sum_{m\in\mathcal{M}}\mathrm{occ}(m)\right) \quad .$$

Adding the $O(n\log\Sigma)$ cost for the suffix tree and the LCA ancestor data structure of Section 3.1, we obtain:

▶ **Theorem 5.** *Given a sequence S of n objects over an alphabet Σ and two integers $q>1$ and $k\geq 0$, a motif trie containing the maximal motifs $\mathcal{M}\subseteq M_{qk}$ and their occurrences $\mathrm{occ}(m)$ in S for $m\in\mathcal{M}$ can be built in time and space $O\Big(n(k+\log\Sigma)+(k+1)^3\times\sum_{m\in\mathcal{M}}\mathrm{occ}(m)\Big)$.*

5 Implementing Generate(u)

We now show how to implement Generate(u) within the time bounds stated by Lemma 4. The idea is as follows. We first obtain E_u, which is an array storing the occurrences in L_u, sorted lexicographically according to the suffix associated with each occurrence. We can then show that there is a bijection between the children of u and a set of maximal intervals in E_u. By exploiting the properties of these intervals, we are able to find them efficiently through a number of scans of E_u. The bijection implies that we thus efficiently obtain the new children of u.

5.1 Nodes of the Motif Trie as Maximal Intervals

The key point in the efficient implementation of the oracle Generate(u) is to relate each node u and its future children $u_1,\ldots,u_d$ labeled by solid blocks $b_1,\ldots,b_d$, respectively, to some suitable intervals that represent their occurrence lists $L_u, L_{u_1},\ldots,L_{u_d}$. Though the idea of using intervals for representing trie nodes is not new (e. g. in [2]), we use intervals to expand the trie rather than merely representing its nodes. Not all intervals generate children as not all solid blocks that extend u necessarily generate a child. Also, some of the solid blocks $b_1, \ldots, b_d$ can be prefixes of each other and one of the intervals can be the empty string ε. To select them carefully, we need some definitions and properties.

Extensions. For a position $p \in L_u$, we define its *extension* as the suffix $\text{ext}(p, u) = S[p + |u| + 1, n]$ that starts at the position after p with an offset equivalent to skipping the prefix matching u plus one symbol (for the don't care). We may write $\text{ext}(p)$, omitting the motif u if it is clear from the context. We also say that the *skipped characters* $\text{skip}(p)$ at position $p \in L_u$ are the $d = \text{dc}(u) + 2$ characters in S that specialize u into its occurrence p: formally, $\text{skip}(p) = \langle c_0, c_1, \ldots, c_{d-1} \rangle$ where $c_0 = S[p-1]$, $c_{d-1} = S[p + |u|]$, and $c_i = S[p + j_i - 1]$, for $1 \leq i \leq d-2$, where $u[j_i] = \star$ is the ith don't care in u.

We denote by E_u the list L_u sorted using as keys the integers for $\text{ext}(p)$ where $p \in L_u$. (We recall from Section 3.1 that the suffixes are represented in the alphabet Π, and thus $\text{ext}(p)$ can be seen as an integer in Π.) By Lemma 2 consecutive positions in E_u share common prefixes of their extensions. Lemma 6 below states that these prefixes are the candidates for being correct edge labels for expanding u in the trie.

▶ **Lemma 6.** *Let u_i be a child of node u, b_i be the label of edge (u, u_i), and $p \in L_u$ be an occurrence position. If position $p \in L_{u_i}$ then b_i is a prefix of $\text{ext}(p, u)$.*

Intervals. Lemma 6 states a necessary condition, so we have to filter the candidate prefixes of the extensions. We use the following notion of intervals to facilitate this task. We call $I \subseteq E_u$ an *interval* of E_u if I contains consecutive entries of E_u. We write $I = [i, j]$ if I covers the range of indices from i to j in E_u. The *longest common prefix* of an interval is defined as $\text{LCP}(I) = \min_{p_1, p_2 \in I} \mathit{lcp}(\text{ext}(p_1), \text{ext}(p_2))$, which is a solid block in Π as discussed at the end of Section 3.1. By Lemma 2, $\text{LCP}(I) = \mathit{lcp}(\text{ext}(E_u[i]), \text{ext}(E_u[j]))$ can be computed in $O(1)$ time, where $E_u[i]$ is the first and $E_u[j]$ the last element in $I = [i, j]$.

Maximal Intervals. An interval $I \subseteq E_u$ is *maximal* if *(1)* there are at least q positions in I (i. e. $|I| \geq q$), *(2)* motif u cannot be specialized with the skipped characters in $\text{skip}(p)$ where $p \in I$, and *(3)* any other interval $I' \subseteq E_u$ that strictly contains I has a shorter common prefix (i. e. $|\text{LCP}(I')| < |\text{LCP}(I)|$ for $I' \supset I$) [5]. We denote by $\mathcal{I}_u$ the *set of all maximal intervals* of E_u, and show that $\mathcal{I}_u$ form a tree covering of E_u. A similar lemma for intervals over the LCP array of a suffix tree was given in [2].

▶ **Lemma 7.** *Let $I_1, I_2 \in \mathcal{I}_u$ be two maximal intervals, where $I_1 \neq I_2$ and $|I_1| \leq |I_2|$. Then either I_1 is contained in I_2 with a longer common prefix (i. e. $I_1 \subset I_2$ and $|\text{LCP}(I_1)| > |\text{LCP}(I_2)|$) or the intervals are disjoint (i. e. $I_1 \cap I_2 = \emptyset$).*

The next lemma establishes a useful bijection between maximal intervals $\mathcal{I}_u$ and children of u, motivating why we use intervals to expand the motif trie.

▶ **Lemma 8.** *Let u_i be a child of a node u. Then the occurrence list L_{u_i} is a permutation of a maximal interval $I \subseteq \mathcal{I}_u$, and vice versa. The label on edge (u, u_i) is the solid block $b_i = \text{LCP}(I)$. No other children or maximal intervals have this property with u_i or I.*

[5] In the full version we show that condition *(2)* is needed to avoid the enumeration of either motifs from $M_{qk} \setminus \mathcal{M}$ or duplicates from $\mathcal{M}$.

5.2 Exploiting the Properties of Maximal Intervals

We now use the properties shown above to implement the oracle GENERATE(u), resulting in Lemma 4. Observe that the task of GENERATE(u) can be equivalently seen by Lemma 8 as the task of finding all maximal intervals $\mathcal{I}_u$ in E_u, where each interval $I \in \mathcal{I}_u$ corresponds exactly to a distinct child u_i of u. We describe three main steps, where the first takes $O(\text{sort}(L_u) + (k+1) \cdot |L_u|)$ time, and the others require $O(\sum_{i=1}^{d} |L_{u_i}|)$ time. The interval $I = E_u$ corresponding to the solid block ε is trivial to find, so we focus on the rest. We assume $\text{dc}(u) < k$, as otherwise we are already done with u.

Step 1. Sort occurrences and find maximal runs of skipped characters. We perform a radix-sort of L_u using the extensions as keys, seen as integers from Π, thus obtaining array E_u. To facilitate the task of checking condition *(2)* for the maximality of intervals, we compute for each index $i \in E_u$ the smallest index $R(i) > i$ in E_u such that motif u cannot be specialized with the skipped characters in $\text{skip}(E_u[j])$ where $j \in [i, R(i)]$. That is, there are at least two different characters from Σ hidden by each of the skipped characters in the interval. (If $R(i)$ does not exist, we do not create $[i, R(i)]$.) We define $|P_I|$ as the minimum number of different characters covered by each skipped character in interval I, and note that $|P_{[i,R(i)]}| \geq 2$ by definition.

To do so we first find, for each skipped character position, all indices where a maximal run of equal characters end: $R(i)$ is the maximum indices for the given i. This helps us because for any index i inside such a block of equal characters, $R(i)$ must be on the right of where the block ends (otherwise $[i, R(i)]$ would cover only one character in that block). Using this to calculate $R(i)$ for all indices $i \in E_u$ from left to right, we find each answer in time $O(k+1)$, and $O((k+1) \cdot |E_u|)$ total time. We denote by $\mathcal{R}$ the set of intervals $[i, R(i)]$ for $i \in E_u$.

▶ **Lemma 9.** *For any maximal interval $I \equiv [i,j] \in \mathcal{I}_u$, there exists $R(i) \leq j$, and thus $[i, R(i)]$ is an initial portion of I.*

Step 2. Find maximal intervals with handles. We want to find all maximal intervals covering each position of E_u. To this end, we introduce *handles.* For each $p \in E_u$, its *interval domain* $D(p)$ is the set of intervals $I' \subset E_u$ such that $p \in I'$ and $|P_{I'}| \geq 2$. We let ℓ_p be the length of the longest shared solid block prefix b_i over $D(p)$, namely, $\ell_p = \max_{I' \in D(p)} |\text{LCP}(I')|$. For a maximal interval $I \subseteq \mathcal{I}_u$, if $|\text{LCP}(I)| = \ell_p$ for some $p \in I$ we call p a *handle* on I. Handles are relevant for the following reason.

▶ **Lemma 10.** *For each maximal interval $I \subseteq \mathcal{I}_u$, either there is a handle $p \in E_u$ on I, or I is fully covered by ≥ 2 adjacent maximal intervals with handles.*

Let $\mathcal{H}_u$ denote the set of maximal intervals with handles. We now show how to find the set $\mathcal{H}_u$ among the intervals of E_u. Observe that for each occurrence $p \in E_u$, we must find the interval I' with the largest $\text{LCP}(I')$ value among all intervals containing p.

From the definition, a handle on a maximal interval I' requires $|P_{I'}| \geq 2$, which is exactly what the intervals in $\mathcal{R}$ satisfy. As the LCP value can only drop when extending an interval, these are the only candidates for maximal intervals with handles. Note that from Lemma 9, $\mathcal{R}$ contains a prefix for all of the (not expanded) maximal intervals because it has all intervals from left to right obeying the conditions on length and skipped character conditions. Furthermore, $|\mathcal{R}| = O(|E_u|)$, since only one $R(i)$ is calculated for each starting position. Among the intervals $[i, R(i)] \in \mathcal{R}$, we will now show how to find those with maximum LCP (i. e. where the LCP value equals ℓ_p) for all p.

We use an idea similar to that used in Section 3.3 to filter maximal motifs from the right-maximal motifs. We sort the intervals $I' = [i, R(i)] \in \mathcal{R}$ in decreasing lexicographic order according to the pairs $\langle |\operatorname{LCP}(I')|, -i\rangle$ (i. e. decreasing LCP values but increasing indices i), to obtain the sequence $\mathcal{C}$. Thus, if considering the intervals left to right in $\mathcal{C}$, we consider intervals with larger LCP values from left to right in S before moving to smaller LCP values.

Consider an interval $I_p = [i, R(i)] \in \mathcal{C}$. The idea is that we determine if I_p has already been added to $\mathcal{H}_u$ by some previously processed handled maximal interval. If not, we expand the interval (making it maximal) and add it to $\mathcal{H}_u$, otherwise I_p is discarded. When $\mathcal{C}$ is fully processed, all occurrences in E_u are covered by maximal intervals with handles.

First, since maximal intervals must be fully contained in each other (from Lemma 7), we determine if $I_p = [i, R(i)] \in \mathcal{C}$ is already fully covered by previously expanded intervals (with larger LCP values) – if not, we must expand I_p. Clearly, if either i or $R(i)$ is not included in any previous expansions, we must expand I_p. Otherwise, if both i and $R(i)$ is part of a single previous expansion $I_q \in \mathcal{C}$, I_p should not be expanded. If i and $R(i)$ is part of two different expansions I_q and I_r we compare their extent values: I_p must be expanded if some $p \in I_p$ is not covered by either I_q or I_r. To enable these checks we mark each $j \in E_u$ with the longest processed interval that contains it (during the expansion procedure below).

Secondly, to expand I_p maximally to the left and right, we use pairwise *lcp* queries on the border of the interval. Let $a \in I_p$ be a border occurrence and $b \notin I_p$ be its neighboring occurrence in E_u (if any, otherwise it is trivial). When $|\mathit{lcp}(a,b)| < |\operatorname{LCP}(I_p)|$, the interval cannot be expanded to span b. When the expansion is completed, I_p is a maximal interval and added to $\mathcal{H}_u$. As previously stated, all elements in I_p are marked as being part of their longest covering handled maximal interval by writing I_p on each of its occurrences.

Step 3. Find composite maximal intervals covered by maximal intervals with handles. From Lemma 10, the only remaining type of intervals are composed of maximal intervals with handles from the set $\mathcal{H}_u$. A *composite maximal interval* must be the union of a sequence of adjacent maximal intervals with handles. We find these as follows. We order $\mathcal{H}_u$ according to the starting position and process it from left to right in a greedy fashion, letting $I_h \in \mathcal{H}_u$ be one of the previously found maximal intervals with handles. Each interval I_h is responsible for generating exactly the composite maximal intervals where the sequence of covering intervals starts with I_h (and which contains a number of adjacent intervals on the right). Let $I'_h \in \mathcal{H}_u$ be the interval adjacent on the right to I_h, and create the composed interval $I_c = I_h + I'_h$ (where $+$ indicates the concatenation of consecutive intervals). To ensure that a composite interval is new, we check as in Step 2 that there is no previously generated maximal interval I with $|\operatorname{LCP}(I)| = |\operatorname{LCP}(I_c)|$ such that $I_c \subseteq I$. This is correct since if there is such an interval, it has already been fully expanded by a previous expansion (of composite intervals or a handled interval). Furthermore, if there is such an interval, all intervals containing I_c with shorter longest common prefixes have been taken care of, since from Lemma 7 maximal intervals cannot straddle each other. If I_c is new, we know that we have a new maximal composite interval and can continue expanding it with adjacent intervals. If the length of the longest common prefix of the expanded interval changes, we must perform the previous check again (and add the previously expanded composite interval to $\mathcal{I}_u$).

By analyzing the algorithm described, one can prove the following two lemmas showing that the motif trie is generated correctly. While Lemma 11 states that ε-extensions may be generated (i. e. a sequence of $\star$ symbols may be added to suffixes of maximal motifs), a simple bottom-up cleanup traversal of T is enough to remove these.

▶ **Lemma 11** (Soundness). *Each motif stored in T is a prefix or an ε-extension of some suffix of a maximal motif (encoded using alphabet Π and stored in T).*

▶ **Lemma 12** (Completeness). *If $m \in \mathcal{M}$, T stores m and its suffixes.*

References

1 Mohamed Ibrahim Abouelhoda and Moustafa Ghanem. String mining in bioinformatics. In *Scientific Data Mining and Knowledge Discovery*, pages 207–247. Springer, 2010.

2 Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *JDA*, 2(1):53–86, 2004.

3 Alberto Apostolico, Matteo Comin, and Laxmi Parida. Bridging lossy and lossless compression by motif pattern discovery. In *General Theory of Information Transfer and Combinatorics*, pages 793–813. Springer, 2006.

4 Hiroki Arimura and Takeaki Uno. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. *JCO*, 2007.

5 Brenda S Baker. On finding duplication and near-duplication in large software systems. In *Proc. 2nd WCRE*, pages 86–95, 1995.

6 Sergey Brin, James Davis, and Héctor García-Molina. Copy detection mechanisms for digital documents. *SIGMOD Rec.*, 24(2):398–409, May 1995.

7 Chia-Hui Chang, Chun-Nan Hsu, and Shao-Cheng Lui. Automatic information extraction from semi-structured web pages by pattern discovery. *Decis Support Syst*, 34(1):129–147, 2003.

8 Xin Chen, Brent Francia, Ming Li, Brian Mckinnon, and Amit Seker. Shared information and program plagiarism detection. *IEEE Trans Inf Theory*, 50(7):1545–1551, 2004.

9 Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

10 Maria Federico and Nadia Pisanti. Suffix tree characterization of maximal motifs in biological sequences. *Theor. Comput. Sci.*, 410(43):4391–4401, 2009.

11 Roberto Grossi, Andrea Pietracaprina, Nadia Pisanti, Geppino Pucci, Eli Upfal, and Fabio Vandin. MADMX: A strategy for maximal dense motif extraction. *J. Comp. Biol.*, 18(4):535–545, 2011.

12 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

13 Nizar R. Mabroukeh and Christie I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM CSUR*, 43(1):3, 2010.

14 Edward M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April 1976.

15 L. Parida, I. Rigoutsos, and D. E. Platt. An output-sensitive flexible pattern discovery algorithm. In *Proc. 12th CPM*, pages 131–142, 2001.

16 Laxmi Parida, Isidore Rigoutsos, Aris Floratos, Daniel E. Platt, and Yuan Gao. Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm. In *Proc. 11th SODA*, pages 297–308, 2000.

17 Lukáš Pichl, Takuya Yamano, and Taisei Kaizoji. On the symbolic analysis of market indicators with the dynamic programming approach. In *Advances in Neural Networks-ISNN*, pages 432–441. Springer, 2006.

18 Isidore Rigoutsos and Tien Huynh. Chung-Kwei: a Pattern-discovery-based System for the Automatic Identification of Unsolicited E-mail Messages. In *CEAS*, 2004.

19 Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Proc. 3rd LATIN*, pages 374–390. Springer, 1998.

20 Reza Sherkat and Davood Rafiei. Efficiently evaluating order preserving similarity queries over historical market-basket data. In *Proc. 22nd ICDE*, pages 19–19, 2006.

21 Esko Ukkonen. Maximal and minimal representations of gapped and non-gapped motifs of a string. *Theor. Comput. Sci.*, 410(43):4341–4349, 2009.

Robust Proximity Search for Balls Using Sublinear Space*

Sariel Har-Peled[1] **and Nirman Kumar**[2]

1 **Department of Computer Science**
University of Illinois
`sariel@uiuc.edu`

2 **Department of Computer Science**
University of Illinois
`nkumar5@uiuc.edu`

Abstract

Given a set of n disjoint balls $b_1, \ldots, b_n$ in $\mathbb{R}^d$, we provide a data structure, of near linear size, that can answer $(1 \pm \varepsilon)$-approximate kth-nearest neighbor queries in $O(\log n + 1/\varepsilon^d)$ time, where k and ε are provided at query time. If k and ε are provided in advance, we provide a data structure to answer such queries, that requires (roughly) $O(n/k)$ space; that is, the data structure has sublinear space requirement if k is sufficiently large.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Approximate Nearest neighbors, algorithms, data structures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.315

1 Introduction

The *nearest neighbor* problem is a fundamental problem in Computer Science [18, 1]. Here, one is given a set of points P, and given a query point q one needs to output the nearest point in P to q. There is a trivial $O(n)$ algorithm for this problem. Typically the set of data points is fixed, while different queries keep arriving. Thus, one can use preprocessing to facilitate a faster query. There are several applications of nearest neighbor search in computer science including pattern recognition, information retrieval, vector compression, computational statistics, clustering, data mining and learning among many others, see for instance the survey by Clarkson [10] for references. If one is interested in guaranteed performance and near linear space, there is no known way to solve this problem efficiently (i. e., logarithmic query time) for dimension $d > 2$, while using near linear space for the data structure.

In light of the above, major effort has been devoted to develop approximation algorithms for nearest neighbor search [6, 17, 10, 13]. In the $(1+\varepsilon)$-*approximate nearest neighbor* problem, one is additionally given an approximation parameter $\varepsilon > 0$ and one is required to find a point $\mathsf{u} \in \mathsf{P}$ such that $\mathsf{d}(\mathsf{q}, \mathsf{u}) \leq (1+\varepsilon)\mathsf{d}(\mathsf{q}, \mathsf{P})$. In d dimensional Euclidean space, one can answer ANN queries in $O(\log n + 1/\varepsilon^{d-1})$ time using linear space [6, 12]. Unfortunately, the constant hidden in the O notation is exponential in the dimension (and this is true for all bounds mentioned in this paper), and specifically because of the $1/\varepsilon^{d-1}$ in the query time, this approach is only efficient in low dimensions. Interestingly, for this data structure,

* Work on this paper was partially support by NSF AF awards CCF-0915984 and CCF-1217462.

the approximation parameter ε need not be specified during the construction, and one can provide it during the query. An alternative approach is to use Approximate Voronoi Diagrams (AVD), introduced by Har-Peled [11], which is a partition of space into regions of low total complexity, with a representative point for each region, that is an ANN for any point in the region. In particular, Har-Peled showed that there is such a decomposition of size $O\bigl((n/\varepsilon^d)\log^2 n\bigr)$, see also [13]. This allows ANN queries to be answered in $O(\log n)$ time. Arya and Malamatos [2] showed how to build AVDs of linear complexity (i. e., $O(n/\varepsilon^d)$). Their construction uses WSPD (Well Separated Pairs Decomposition) [8]. Further trade-offs between query time and space usage for AVDs were studied by Arya *et al.* [4].

A more general problem is the k-nearest neighbors problem where one is interested in finding the k points in P nearest to the query point q. This is widely used in classification, where the majority label is used to label the query point. A restricted version is to find only the kth-nearest neighbor. This problem and its approximate version have been considered in [3, 14].

Recently, the authors [14] showed that one can compute a (k,ε)-AVD that $(1+\varepsilon)$-approximates the distance to the kth nearest neighbor, and surprisingly, requires $O(n/k)$ space; that is, sublinear space if k is sufficiently large. For example, for the case $k=\Omega(\sqrt{n})$, which is of interest in practice, the space required is only $O(\sqrt{n})$. Such ANN is of interest when one is worried that there is noise in the data, and thus one is interested in the distance to the kth NN which is more robust and noise resistant. Alternatively, one can think about such data structures as enabling one to summarize the data in a way that still facilitates meaningful proximity queries.

In this paper we consider a generalization of the kth-nearest neighbor problem. Here, we are given a set of n disjoint balls in $\mathbb{R}^d$ and we want to preprocess them, so that given a query point we can find approximately the kth closest ball. The distance of a query point to a ball is defined as the distance to its boundary if the point is outside the ball or 0 otherwise. Clearly, this problem is a generalization of the kth-nearest neighbor problem by viewing points as balls of radius 0. Algorithms for the kth-nearest neighbor for points, do not extend in a straightforward manner to this problem because the distance function is no longer a metric. Indeed, there can be two very far off points both very close to a single ball, and thus the triangle inequality does not hold. The problem of finding the closest ball can also be modeled as a problem of approximating the minimization diagram of a set of functions; here, a function would correspond to the distance from one of the given balls. There has been some recent work by the authors on this topic, see [15], where a fairly general class of functions admits a near-linear sized data structure permitting a logarithmic time query for the problem of approximating the minimization diagram. However, the problem that we consider in this paper does not fall under the framework of [15]. The technical assumptions of [15] mandate that the set of points which form the 0-sublevel set of a distance function, i. e., the set of points at which the distance function is 0 is a single point (or an empty set). This is not the case for the problem we consider here. Also, we are interested in the more general kth-nearest neighbor problem, while [15] only considers the nearest-neighbor problem, i. e., $k=1$.

We first show how to preprocess the set of balls into a data structure requiring space $O(n)$, in $O(n\log n)$ time, so that given a query point q, a number $1\le k\le n$ and $\varepsilon>0$, one can compute a $(1\pm\varepsilon)$-approximate kth closest ball in time $O(\log n+\varepsilon^{-d})$. If both k and ε are available during preprocessing, one can preprocess the balls into a (k,ε)-AVD, using $O(\frac{n}{k\varepsilon^d}\log(1/\varepsilon))$ space, so that given a query point q, a (k,ε)-ANN closest ball can be computed, in $O(\log(n/k)+\log(1/\varepsilon))$ time.

Paper Organization. In Section 2, we define the problem, list some assumptions, and introduce notations. In Section 3, we set up some basic data structures to answer approximate range counting queries for balls. In Section 4, we present the data structure, query algorithm and proof of correctness for our data structure which can compute $(1 \pm \varepsilon)$-approximate kth-nearest neighbors of a query point when k, ε are only provided during query time. In Section 5 we present approximate quorum clustering, see [9, 14], for a set of disjoint balls. Using this, in Section 6, we present the (k, ε)-AVD construction. We conclude in Section 7.

2 Problem definition and notation

We are given a set of disjoint[1] balls $\mathcal{B} = \{b_1, \ldots, b_n\}$, where $b_i = \mathsf{b}(\mathsf{c}_i, \mathsf{r}_i)$, for $i = 1, \ldots, n$. Here $\mathsf{b}(\mathsf{c}, \mathsf{r}) \subseteq \mathbb{R}^d$ denotes the (closed) ball with center c and radius $\mathsf{r} \geq 0$. Additionally, we are given an approximation parameter $\varepsilon \in (0, 1)$. For a point $\mathsf{q} \in \mathbb{R}^d$, the *distance* of q to a ball $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ is $\mathsf{d}(\mathsf{q}, b) = \max\Big(\|\mathsf{q} - \mathsf{c}\| - \mathsf{r}, 0 \Big)$.

▶ **Observation 1.** *For two balls $b_1 \subseteq b_2 \subseteq \mathbb{R}^d$, and any point $\mathsf{q} \in \mathbb{R}^d$, we have $\mathsf{d}(\mathsf{q}, b_1) \geq \mathsf{d}(\mathsf{q}, b_2)$.*

The *kth-nearest neighbor distance* of q to $\mathcal{B}$, denoted by $\mathsf{d}_k(\mathsf{q}, \mathcal{B})$, is the kth smallest number in $\mathsf{d}(\mathsf{q}, b_1), \ldots, \mathsf{d}(\mathsf{q}, b_n)$. Similarly, for a given set of points P, $\mathsf{d}_k(\mathsf{q}, \mathsf{P})$ denotes the kth-nearest neighbor distance of q to P.

We aim to build a data structure to answer $(1 \pm \varepsilon)$-approximate kth-nearest neighbor (i. e., (k, ε)-*ANN*) queries, where for any query point $\mathsf{q} \in \mathbb{R}^d$ one needs to output a ball $b \in \mathcal{B}$ such that, $(1 - \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. There are different variants depending on whether ε and k are provided with the query or in advance.

We use *cube* to denote a set of the form $[a_1, a_1 + \ell] \times [a_2, a_2 + \ell] \times \ldots \times [a_d, a_d + \ell] \subseteq \mathbb{R}^d$, where $a_1, \ldots, a_d \in \mathbb{R}$ and $\ell \geq 0$ is the side length of the cube.

▶ **Observation 2.** *For any set of balls $\mathcal{B}$, the function $\mathsf{d}_k(\mathsf{q}, \mathcal{B})$ is a 1-Lipschitz function; that is, for any two points u, v, we have that $\mathsf{d}_k(\mathsf{u}, \mathcal{B}) \leq \mathsf{d}_k(\mathsf{v}, \mathcal{B}) + \|\mathsf{u} - \mathsf{v}\|$.*

▶ **Assumption 3.** *We assume all the balls are contained inside the cube $\Big[1/2 - \delta, 1/2 + \delta\Big]^d$, which can be ensured by translation and scaling (which preserves order of distances), where $\delta = \varepsilon/4$. As such, we can ignore queries outside the unit cube $[0, 1]^d$, as any input ball is a valid answer in this case.*

For a real positive number x and a point $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d) \in \mathbb{R}^d$, define $\mathsf{G}_x(\mathsf{p})$ to be the grid point $(\lfloor \mathsf{p}_1/x \rfloor x, \ldots, \lfloor \mathsf{p}_d/x \rfloor x)$. The number x is the *width* or *side length* of the *grid* G_x. The mapping G_x partitions $\mathbb{R}^d$ into cubes that are called grid *cells*.

▶ **Definition 4.** A cube is a *canonical cube* if it is contained inside the unit cube $U = [0, 1]^d$, it is a cell in a grid G_r, and r is a power of two (i. e., it might correspond to a node in a quadtree having $[0, 1]^d$ as its root cell). We will refer to such a grid G_r as a *canonical grid*. Note that all the cells corresponding to nodes of a compressed quadtree are canonical.

[1] Our data structure and algorithm work for the more general case where the balls are interior disjoint, where we define the interior of a "point ball", i. e., a ball of radius 0, as the point itself. This is not the usual topological definition.

▶ **Definition 5.** Given a set $b \subseteq \mathbb{R}^d$, and a parameter $\delta > 0$, let $\mathsf{G}_{\approx}(b, \delta)$ denote the set of canonical grid cells of side length $2^{\lfloor \log_2 \delta \mathrm{diam}(b)/\sqrt{d} \rfloor}$, that intersect b, where $\mathrm{diam}(b) = \max_{\mathsf{p},\mathsf{u} \in b} \|\mathsf{p} - \mathsf{u}\|$ denotes the *diameter* of b. Clearly, the diameter of any grid cell of $\mathsf{G}_{\approx}(b, \delta)$, is at most $\delta \mathrm{diam}\,(b)$. Let $\mathsf{G}_{\approx}(b) = \mathsf{G}_{\approx}(b, 1)$. It is easy to verify that $|\mathsf{G}_{\approx}(b)| = O(1)$. The set $\mathsf{G}_{\approx}(b)$ is the *grid approximation* to b.

Let $\mathcal{B}$ be a family of balls in $\mathbb{R}^d$. Given a set $X \subseteq \mathbb{R}^d$, let

$$\mathcal{B}(X) = \left\{ b \in \mathcal{B} \;\middle|\; b \cap X \neq \emptyset \right\}$$

denote the set of all balls in $\mathcal{B}$ that intersect X.

For two compact sets $X, Y \subseteq \mathbb{R}^d$, $X \preceq Y$ if and only if $\mathrm{diam}(X) \leq \mathrm{diam}(Y)$. For a set X and a set of balls $\mathcal{B}$, let $\mathcal{B}_{\succeq}(X) = \left\{ b \in \mathcal{B} \;\middle|\; b \cap X \neq \emptyset \text{ and } b \succeq X \right\}$. Let c_d denote the maximum number of pairwise disjoint balls of radius at least r, that may intersect a given ball of radius r in $\mathbb{R}^d$. Clearly, we have $|\mathcal{B}_{\succeq}(b)| \leq \mathsf{c}_d$ for any ball b. The proof of the following lemma appears in the full version [16].

▶ **Lemma 6.** $2 \leq \mathsf{c}_d \leq 3^d$ *for all* d.

▶ **Definition 7.** For a parameter $\delta \geq 0$, a function $f : \mathbb{R}^+ \to \mathbb{R}^+$ is *δ-monotonic*, if for every $x \geq 0$, $f(x/(1+\delta)) \leq f(x)$.

3 Approximate range counting for balls

▶ **Data-structure 8.** For a given set of disjoint balls $\mathcal{B} = \{b_1, \ldots, b_n\}$ in $\mathbb{R}^d$, we build the following data structure, that is useful in performing several of the tasks at hand.

(A) **Store balls in a (compressed) quadtree.** For $i = 1, 2, \ldots, n$, let $G_i = \mathsf{G}_{\approx}(b_i)$, and let $G = \bigcup_{i=1}^{n} G_i$ denote the union of these cells. Let $\mathcal{T}$ be a compressed quadtree decomposition of $[0,1]^d$, such that all the cells of G are cells of $\mathcal{T}$. We preprocess $\mathcal{T}$ to answer point location queries for the cells of G. This takes $O(n \log n)$ time, see [12].

(B) **Compute list of "large" balls intersecting each cell.** For each node u of $\mathcal{T}$, there is a list of balls registered with it. Formally, *register* a ball b_i with all the cells of G_i. Clearly, each ball is registered with $O(1)$ cells, and it is easy to see that each cell has $O(1)$ balls registered with it, since the balls are disjoint.
Next, for a cell $\square$ in $\mathcal{T}$ we compute a list storing $\mathcal{B}_{\succeq}(\square)$, and these balls are *associated* with this cell. These lists are computed in a top-down manner. To this end, propagate from a node u its list $\mathcal{B}_{\succeq}(\square)$ (which we assume is already computed) down to its children. For a node receiving such a list, it scans it, and keep only the balls that intersect its cell (adding to this list the balls already registered with this cell). For a node $\nu \in \mathcal{T}$, let $\mathcal{B}_\nu$ be this list.

(C) **Build compressed quadtree on centers of balls.** Let $\mathcal{C}$ be the set of centers of the balls of $\mathcal{B}$. Build, in $O(n \log n)$ time, a compressed quadtree $\mathcal{T}_{\mathcal{C}}$ storing $\mathcal{C}$.

(D) **ANN for centers of balls.** Build a data structure $\mathcal{D}$, for answering 2-approximate k-nearest neighbor distances on $\mathcal{C}$, the set of centers of the balls, see [14], where k and ε are provided with the query. The data structure $\mathcal{D}$, returns a point $\mathsf{c} \in \mathcal{C}$ such that, $\mathsf{d}_k(\mathsf{q}, \mathcal{C}) \leq \mathsf{d}(\mathsf{q}, \mathsf{c}) \leq 2\mathsf{d}_k(\mathsf{q}, \mathcal{C})$.

(E) **Answering approximate range searching for the centers of balls.** Given a query ball $b_{\mathsf{q}} = \mathsf{b}(\mathsf{q}, x)$ and a parameter $\delta > 0$, one can, using $\mathcal{T}_{\mathcal{C}}$, report (approximately), in $O(\log n + 1/\delta^d)$ time, the points in $b_{\mathsf{q}} \cap \mathcal{C}$. Specifically, the query process computes $O(1/\delta^d)$ sets of points, such that their union X, has the property that $b_{\mathsf{q}} \cap \mathcal{C} \subseteq X \subseteq$

$(1+\delta)b_{\mathsf{q}} \cap \mathcal{C}$, where $(1+\delta)b_{\mathsf{q}}$ is the scaling of b_{q} by a factor of $1+\delta$ around its center. Indeed, compute the set $\mathsf{G}_{\approx}\bigl(b_{\mathsf{q}}\bigr)$, and then using cell queries in $\mathcal{T}_{\mathcal{C}}$ compute the corresponding cells (this takes $O(\log n)$ time). Now, descend to the relevant level of the quadtree to all the cells of the right size, that intersect b_{q}. Clearly, the union of points stored in their subtrees are the desired set. This takes overall $O(\log n + 1/\delta^d)$ time. A similar data structure for approximate range searching is provided by Arya and Mount [5], and our description above is provided for the sake of completeness.

Overall, it takes $O(n \log n)$ time to build this data structure.

We denote the collection of data structures above by $\mathcal{DS}_8$ and where necessary, specific functionality it provides, say for finding the large balls intersecting a cell, by $\mathcal{DS}_8$ (2).

3.1 Approximate range counting among balls

We need the ability to answer approximate range counting queries on a set of disjoint balls. Specifically, given a set of disjoint balls $\mathcal{B}$, and a query ball b, the target is to compute the size of the set $b \cap \mathcal{B} = \left\{ b' \in \mathcal{B} \;\middle|\; b' \cap b \neq \emptyset \right\}$. To make this query computationally fast, we allow an approximation. More precisely, for a ball b a set $\widetilde{b}$ is a $(1+\delta)$-*ball* of b, if $b \subseteq \widetilde{b} \subseteq (1+\delta)b$, where $(1+\delta)b$ is the $(1+\delta)$-scaling of b around its center. The purpose here, given a query ball b, is to compute the size of the set $\widetilde{b} \cap \mathcal{B}$ for some $(1+\delta)$-ball $\widetilde{b}$ of b.

The proofs of the following two lemmas appear in the full version [16].

▶ **Lemma 9.** *Given a compressed quadtree $\mathcal{T}$ of size n, a convex set X, and a parameter $\delta > 0$, one can compute the set of nodes in $\mathcal{T}$, that realizes $\mathsf{G}_{\approx}(X, \delta)$ (see Defnition 5), in $O\bigl(\log n + 1/\delta^d\bigr)$ time. Specifically, this outputs a set X_N of nodes, of size $O\bigl(1/\delta^d\bigr)$, such that their cells intersect $\mathsf{G}_{\approx}(X, \delta)$, and their parents cell diameter is larger than $\delta \mathrm{diam}(X)$. Note that the cells in X_N might be significantly larger if they are leaves of $\mathcal{T}$.*

▶ **Lemma 10.** *Let X be any convex set in $\mathbb{R}^d$, and let $\delta > 0$ be a parameter. Using $\mathcal{DS}_8$, one can compute, in $O\bigl(\log n + 1/\delta^d\bigr)$ time, all the balls of $\mathcal{B}$ that intersect X, with diameter $\geq \delta \mathrm{diam}(X)$.*

3.2 Answering a query

Given a query ball $b_{\mathsf{q}} = \mathsf{b}(\mathsf{q}, x)$, and an approximation parameter $\delta > 0$, our purpose is to compute a number N, such that $\left|\mathcal{B}\Bigl(\mathsf{b}(\mathsf{q}, x)\Bigr)\right| \leq N \leq \left|\mathcal{B}\Bigl(\mathsf{b}(\mathsf{q}, (1+\delta)x)\Bigr)\right|$.

The query algorithm works as follows:

(A) Using Lemma 10, compute a set X of all the balls that intersect b_{q} and are of radius $\geq \delta x/4$.

(B) Using $\mathcal{DS}_8$, compute $O(1/\delta^d)$ cells of $\mathcal{T}_{\mathcal{C}}$ that corresponds to $\mathsf{G}_{\approx}\bigl(b_{\mathsf{q}}(1+\delta/4), \delta/4\bigr)$. Let N' be the total number of points in $\mathcal{C}$ stored in these nodes.

(C) The quantity $N' + |X|$ is almost the desired quantity, except that we might be counting some of the balls of X twice. To this end, let N'' be the number of balls in X with centers in $\mathsf{G}_{\approx}\bigl(b_{\mathsf{q}}(1+\delta/4), \delta/4\bigr)$

(D) Let $N \leftarrow N' + |X| - N''$. Return N.

We only sketch the proof, as the proof is straightforward. Indeed, the union of the cells of $\mathsf{G}_{\approx}\bigl(b_{\mathsf{q}}(1+\delta/4), \delta/4\bigr)$ contains $\mathsf{b}(\mathsf{q}, x(1+\delta/4))$ and is contained in $\mathsf{b}(\mathsf{q}, (1+\delta)x)$. All the balls with radius smaller than $\delta x/4$ and intersecting $\mathsf{b}(\mathsf{q}, x)$ have their centers in cells of $\mathsf{G}_{\approx}\bigl(b_{\mathsf{q}}(1+\delta/4), \delta/4\bigr)$, and their number is computed correctly. Similarly, the "large" balls are computed correctly. The last stage ensures we do not over-count by 1 each large ball that also has its center in $\mathsf{G}_{\approx}\big(b_{\mathsf{q}}(1+\delta/4), \delta/4\big)$. It is also easy to check that $|\mathcal{B}(\mathsf{b}(\mathsf{q}, x))| \leq N \leq |\mathcal{B}(\mathsf{b}(\mathsf{q}, x(1+\delta)))|$. The same result can be used for $x/(1+\delta)$ to get δ-monotonicity of N.

We now analyze the running time. Computing all the cells of $\mathsf{G}_{\approx}\big(b_{\mathsf{q}}(1+\delta/4), \delta/4\big)$ takes $O(\log n + 1/\delta^d)$ time. Computing the "large" balls takes $O\big(\log n + 1/\delta^d\big)$ time. Checking for each large ball if it is already counted by the "small" balls takes $O(1/\delta^d)$ by using a grid. We denote the above query algorithm by **rangeCount**(q, x, δ).

The above implies the following.

▶ **Lemma 11.** *Given a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, it can be preprocessed, in $O(n \log n)$ time, into a data structure of size $O(n)$, such that given a query ball $\mathsf{b}(\mathsf{q}, x)$ and approximation parameter $\delta > 0$, the query algorithm **rangeCount**(q, x, δ) returns, in $O(\log n + 1/\delta^d)$ time, a number N satisfying the following:*

(A) $N \leq |\mathcal{B}(\mathsf{b}(\mathsf{q}, (1+\delta)x))|$,

(B) $|\mathcal{B}(\mathsf{b}(\mathsf{q}, x))| \leq N$, *and*

(C) *for a query ball $\mathsf{b}(\mathsf{q}, x)$ and δ, the number N is δ-monotonic as a function of x, see Defnition 7.*

4 Answering k-ANN queries among balls

4.1 Computing a constant factor approximation to $\mathsf{d}_k(\mathsf{q}, \mathcal{B})$

The proof of the following lemma appears in the full version [16].

▶ **Lemma 12.** *Let $\mathcal{B}$ be a set of disjoint balls in $\mathbb{R}^d$, and consider a ball $b = \mathsf{b}(\mathsf{q}, r)$ that intersects at least k balls of $\mathcal{B}$. Then, among the k nearest neighbors of q from $\mathcal{B}$, there are at least $\max(0, k - \mathsf{c}_d)$ balls of radius at most r. The centers of all these balls are in $\mathsf{b}(\mathsf{q}, 2r)$.*

▶ **Corollary 13.** *Let $\gamma = \min(k, \mathsf{c}_d)$. Then, $\mathsf{d}_{k-\gamma}(\mathsf{q}, \mathcal{C})/2 \leq \mathsf{d}_k(\mathsf{q}, \mathcal{B})$.*

The basic observation is that we only need a rough approximation to the right radius, as using approximate range counting (i. e., Lemma 11), one can improve the approximation.

Let x_i denote the distance of q to the ith closest center in $\mathcal{C}$. Let $d_k = \mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Let i be the minimum index, such that $d_k \leq x_i$. Since $d_k \leq x_k$, it must be that $i \leq k$. There are several possibilities:

(A) If $i \leq k - \mathsf{c}_d$ (i. e., $d_k \leq x_{k-\mathsf{c}_d}$) then, by Lemma 12, the ball $\mathsf{b}(\mathsf{q}, 2d_k)$ contains at least $k - \mathsf{c}_d$ centers. As such, $d_k < x_{k-\mathsf{c}_d} \leq 2d_k$, and $x_{k-\mathsf{c}_d}$ is a good approximation to d_k.

(B) If $i > k - \mathsf{c}_d$, and $d_k \leq 4x_{i-1}$, then x_{i-1} is the desired approximation.

(C) If $i > k - \mathsf{c}_d$, and $d_k \geq x_i/4$, then x_i is the desired approximation.

(D) Otherwise, it must be that $i > k - \mathsf{c}_d$, and $4x_{i-1} < d_k < x_i/4$. Let $b_j = \mathsf{b}(\mathsf{c}_j, \mathsf{r}_j)$ be the jth closest ball to q, for $j = 1, \ldots, k$. It must be that $b_i, \ldots, b_k$ are much larger than $\mathsf{b}(\mathsf{q}, d_k)$. But then, the balls $b_i, \ldots, b_k$ must intersect $\mathsf{b}(\mathsf{q}, x_i/2)$, and their radius is at least $x_i/2$. We can easily compute these big balls using $\mathcal{DS}_8$ (2), and the number of centers of the small balls close to query, and then compute d_k exactly.

We build $\mathcal{DS}_8$ in $O(n \log n)$ time.

First we introduce some notation. For $x \geq 0$, let $N(x)$ denote the number of balls in $\mathcal{B}$ that intersect $\mathsf{b}(\mathsf{q}, x)$; that is $N(x) = \left|\left\{ b \in \mathcal{B} \;\middle|\; b \cap \mathsf{b}(\mathsf{q}, x) \neq \emptyset \right\}\right|$, and $C(x)$ denote the number of centers in $\mathsf{b}(\mathsf{q}, x)$, i. e., $C(x) = |\mathcal{C} \cap \mathsf{b}(\mathsf{q}, x)|$. Also, let $\#(x)$ denote the 2-approximation to the number of balls of $\mathcal{B}$ intersecting $\mathsf{b}(\mathsf{q}, x)$, as computed by Lemma 11; that is $N(x) \leq \#(x) \leq N(2x)$.

We now provide our algorithm to answer a query. We are given a query point $\mathsf{q} \in \mathbb{R}^d$ and a number k.

Using $\mathcal{DS}_8$, compute a 2-approximation for the smallest ball containing $k-i$ centers of $\mathcal{B}$, for $i = 0, \ldots, \gamma$, where $\gamma = \min(k, \mathsf{c}_d)$, and let r_{k-i} be this radius. That is, for $i = 0, \ldots, \gamma$, we have $C(r_{k-i}/2) \leq k - i \leq C(r_{k-i})$. For $i = 0, \ldots, \gamma$, compute $N_{k-i} = \#(r_{k-i})$ (Lemma 11).

Let α be the maximum index such that $N_{k-\alpha} \geq k$. Clearly, α is well defined as $N_k \geq k$. The algorithm is executed in the following steps.

(A) If $\alpha = \gamma$ we return $2r_{k-\gamma}$.

(B) If $\#(r_{k-\alpha}/4) < k$, we return $2r_{k-\alpha}$.

(C) Otherwise, compute all the balls of $\mathcal{B}$ that are of radius at least $r_{k-\alpha}/4$ and intersect the ball $\mathsf{b}(\mathsf{q}, r_{k-\alpha}/4)$, using $\mathcal{DS}_8$ (2). For each such ball b, compute the distance $\zeta = \mathsf{d}(\mathsf{q}, b)$ of q to it. Return 2ζ for the minimum such number ζ such that $\#(\zeta) \geq k$.

The proof of the following lemma appears in the full version [16].

▶ **Lemma 14.** *Given a set of n disjoint balls $\mathcal{B}$ in $\mathbb{R}^d$, one can preprocess them, in $O(n \log n)$ time, into a data structure of size $O(n)$, such that given a query point $\mathsf{q} \in \mathbb{R}^d$, and a number k, one can compute, in $O(\log n)$ time, a number x such that, $x/4 \leq \mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq 4x$.*

We now show how to refine the approximation in the following lemma, whose proof appears in the full version [16].

▶ **Lemma 15.** *Given a set $\mathcal{B}$ of n balls in $\mathbb{R}^d$, it can be preprocessed, in $O(n \log n)$ time, into a data structure of size $O(n)$. Given a query point q, numbers k, x, and an approximation parameter $\varepsilon > 0$, such that $x/4 \leq \mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq 4x$, one can find a ball $b \in \mathcal{B}$ such that, $(1-\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$, in $O\big(\log n + 1/\varepsilon^d\big)$ time.*

4.2 The result

▶ **Theorem 16.** *Given a set of n disjoint balls $\mathcal{B}$ in $\mathbb{R}^d$, one can preprocess them in time $O(n \log n)$ into a data structure of size $O(n)$, such that given a query point $\mathsf{q} \in \mathbb{R}^d$, a number k with $1 \leq k \leq n$ and $\varepsilon > 0$, one can find in time $O\big(\log n + \varepsilon^{-d}\big)$ a ball $b \in \mathcal{B}$, such that, $(1-\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$.*

5 Quorum clustering

We are given a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, and we describe how to compute quorum clustering for them quickly.

Let ξ be some constant. Let $\mathcal{B}_0 = \emptyset$. For $i = 1, \ldots, m$, let $\mathcal{R}_i = \mathcal{B} \setminus (\bigcup_{j=0}^{i-1} \mathcal{B}_j)$, and let $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ be any ball that satisfies,

(A) Λ_i contains $\min(k - \mathsf{c}_d, |\mathcal{R}_i|)$ balls of $\mathcal{R}_i$ completely inside it,

(B) Λ_i intersects at least k balls of $\mathcal{B}$, and

(C) the radius of Λ_i is at most ξ times the radius of the smallest ball satisfying the above conditions.

Next, we remove any $k - \mathsf{c}_d$ balls that are contained in Λ_i from $\mathcal{R}_i$ to get the set $\mathcal{R}_{i+1}$. We call the removed set of balls $\mathcal{B}_i$. We repeat this process till all balls are extracted. Notice that at each step i, we only require that the Λ_i intersects k balls of $\mathcal{B}$ (and not $\mathcal{R}_i$), but that it must contain $k - \mathsf{c}_d$ balls from $\mathcal{R}_i$. Also, the last quorum ball may contain fewer balls. The balls $\Lambda_1, \ldots, \Lambda_m$, are the resulting *$\xi$-approximate quorum clustering.*

5.1 Computing an approximate quorum clustering

▶ **Definition 17.** For a set P of n points in $\mathbb{R}^d$, and an integer ℓ, with $1 \leq \ell \leq n$, let $r_{\mathrm{opt}}(\mathsf{P}, \ell)$ denote the radius of the smallest ball which contains at least ℓ points from P, i. e., $r_{\mathrm{opt}}(\mathsf{P}, \ell) = \min_{\mathsf{q} \in \mathbb{R}^d} \mathsf{d}_\ell(\mathsf{q}, \mathsf{P})$.

Similarly, for a set $\mathcal{R}$ of n balls in $\mathbb{R}^d$, and an integer ℓ, with $1 \leq \ell \leq n$, let $R_{\mathrm{opt}}(\mathcal{R}, \ell)$ denote the radius of the smallest ball which completely contains at least ℓ balls from $\mathcal{R}$.

▶ **Lemma 18** ([14]). *Given a set P of n points in $\mathbb{R}^d$ and integer ℓ, with $1 \leq \ell \leq n$, one can compute, in $O(n \log n)$ time, a sequence of $\lceil n/\ell \rceil$ balls, $\mathsf{o}_1 = \mathsf{b}(\mathsf{u}_1, \psi_1), \ldots, \mathsf{o}_{\lceil n/\ell \rceil} = \mathsf{b}(\mathsf{u}_{\lceil n/\ell \rceil}, \psi_{\lceil n/\ell \rceil})$, such that, for all $i, 1 \leq i \leq \lceil n/\ell \rceil$, we have*

(A) *For every ball o_i, there is an associated subset P_i of $\min(\ell, |\mathsf{Q}_i|)$ points of $\mathsf{Q}_i = \mathsf{P} \setminus (\mathsf{P}_i \cup \ldots \cup \mathsf{P}_{i-1})$, that it covers.*

(B) *The ball $\mathsf{o}_i = \mathsf{b}(\mathsf{u}_i, \psi_i)$ is a 2-approximation to the smallest ball covering $\min(\ell, |\mathsf{Q}_i|)$ points in Q_i; that is, $\psi_i/2 \leq r_{\mathrm{opt}}(\mathsf{Q}_i, \min(\ell, |\mathsf{Q}_i|)) \leq \psi_i$.*

The algorithm to construct an approximate quorum clustering is as follows. We use the algorithm of Lemma 18 with the set of points $\mathsf{P} = \mathcal{C}$, and $\ell = k - \mathsf{c}_d$ to get a list of $m = \lceil n/(k - \mathsf{c}_d) \rceil$ balls $\mathsf{o}_1 = \mathsf{b}(\mathsf{u}_1, \psi_1), \ldots, \mathsf{o}_m = \mathsf{b}(\mathsf{u}_m, \psi_m)$, satisfying the conditions of Lemma 18. Next we use the algorithm of Theorem 16, to compute (k, ε)-ANN distances from the centers $\mathsf{u}_1, \ldots, \mathsf{u}_m$, to the balls of $\mathcal{B}$.

Thus, we get numbers γ_i satisfying, $(1/2)\mathsf{d}_k(\mathsf{u}_i, \mathcal{B}) \leq \gamma_i \leq (3/2)\mathsf{d}_k(\mathsf{u}_i, \mathcal{B})$. Let $\zeta_i = \max(2\gamma_i, 3\psi_i)$, for $i = 1, \ldots, m$. Sort $\zeta_1, \ldots, \zeta_m$ (we assume for the sake of simplicity of exposition that ζ_m, being the radius of the last cluster is the largest number). Suppose the sorted order is the permutation π of $\{1, \ldots, m\}$ (by assumption $\pi(m) = m$). We output the balls $\Lambda_i = \mathsf{b}(\mathsf{u}_{\pi(i)}, \zeta_{\pi(i)})$, for $i = 1, \ldots, m$, as the approximate quorum clustering.

5.2 Correctness

The following lemmas prove the correctness of our approximate quorum clustering algorithm. Their proofs appear in the full version [16].

▶ **Lemma 19.** *Let $\mathcal{B} = \{b_1, \ldots, b_n\}$ be a set of n disjoint balls, where $b_i = \mathsf{b}(\mathsf{c}_i, \mathsf{r}_i)$, for $i = 1, \ldots, n$. Let $\mathcal{C} = \{\mathsf{c}_1, \ldots, \mathsf{c}_n\}$ be the set of centers of these balls. Let $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ be any ball that contains at least ℓ centers from $\mathcal{C}$, for some $2 \leq \ell \leq n$. Then $\mathsf{b}(\mathsf{c}, 3\mathsf{r})$ contains the ℓ balls that correspond to those centers.*

▶ **Lemma 20.** *Let $\mathcal{B} = \{b_1 = \mathsf{b}(\mathsf{c}_1, \mathsf{r}_1), \ldots, b_n = \mathsf{b}(\mathsf{c}_n, \mathsf{r}_n)\}$ be a set of n disjoint balls in $\mathbb{R}^d$. Let $\mathcal{C} = \{\mathsf{c}_1, \ldots, \mathsf{c}_n\}$ be the corresponding set of centers, and let ℓ be an integer with $2 \leq \ell \leq n$. Then, $r_{\mathrm{opt}}(\mathcal{C}, \ell) \leq R_{\mathrm{opt}}(\mathcal{B}, \ell) \leq 3r_{\mathrm{opt}}(\mathcal{C}, \ell)$.*

▶ **Lemma 21.** *The balls $\Lambda_1, \ldots \Lambda_m$ computed above are a 12-approximate quorum clustering of $\mathcal{B}$.*

▶ **Lemma 22.** *Given a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, such that $(k - \mathsf{c}_d)|n$, and a number k with $2\mathsf{c}_d < k \leq n$, in $O(n \log n)$ time, one can output a sequence of $m = n/(k - \mathsf{c}_d)$ balls $\Lambda_1, \ldots, \Lambda_m$, such that*

(A) *For each ball Λ_i, there is an associated subset $\mathcal{B}_i$ of $k - \mathsf{c}_d$ balls of $\mathcal{R}_i = \mathcal{B} \setminus (\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_{i-1})$, that it completely covers.*

(B) *The ball Λ_i intersects at least k balls from $\mathcal{B}$.*

(C) *The radius of the ball Λ_i is at most 12 times that of the smallest ball covering $k - \mathsf{c}_d$ balls of $\mathcal{R}_i$ completely, and intersecting k balls of $\mathcal{B}$.*

Proof. The correctness was proved in Lemma 21. To see the time bound is also easy as the computation time is dominated by the time in Lemma 18, which is $O(n \log n)$. ◀

6 Construction of the sublinear space data structure for (k, ε)-ANN

Here we show how to compute an approximate Voronoi diagram for approximating the kth-nearest ball, that takes $O(n/k)$ space. We assume $k > 2\mathsf{c}_d$ without loss of generality, and we let $m = \lceil n/(k - \mathsf{c}_d)\rceil = O(n/k)$. Here k and ε are prespecified in advance.

6.1 Preliminaries

The following notation was introduced in [14]. A ball b of radius r in $\mathbb{R}^d$, centered at a point c, can be interpreted as a point in $\mathbb{R}^{d+1}$, denoted by $b' = (\mathsf{c}, \mathsf{r})$. For a regular point $\mathsf{p} \in \mathbb{R}^d$, its corresponding image under this transformation is the *mapped* point $\mathsf{p}' = (\mathsf{p}, 0) \in \mathbb{R}^{d+1}$, i. e., we view it as a ball of radius 0 and use the mapping defined on balls. Given point $\mathsf{u} = (\mathsf{u}_1, \ldots, \mathsf{u}_d) \in \mathbb{R}^d$ we will denote its Euclidean norm by $\|\mathsf{u}\|$. We will consider a point $\mathsf{u} = (\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_{d+1}) \in \mathbb{R}^{d+1}$ to be in the product metric of $\mathbb{R}^d \times \mathbb{R}$ and endowed with the product metric norm

$$\|\mathsf{u}\|_{\oplus} = \sqrt{\mathsf{u}_1^2 + \cdots + \mathsf{u}_d^2} + |\mathsf{u}_{d+1}|.$$

It can be verified that the above defines a norm, and for any $\mathsf{u} \in \mathbb{R}^{d+1}$ we have $\|\mathsf{u}\| \leq \|\mathsf{u}\|_{\oplus} \leq \sqrt{2}\,\|\mathsf{u}\|$.

6.2 Construction

The input is a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, and parameters k and ε.

The construction of the data structure is similar to the construction of the kth-nearest neighbor data structure from the authors' paper [14]. We compute, using Lemma 22, a ξ-approximate quorum clustering of $\mathcal{B}$ with $m = n/(k - \mathsf{c}_d) = O(n/k)$ balls, $\Sigma = \{\Lambda_1 = \mathsf{b}(\mathsf{w}_1, \mathsf{x}_1), \ldots, \Lambda_m = \mathsf{b}(\mathsf{w}_m, \mathsf{x}_m)\}$, where $\xi \leq 12$. The algorithm then continues as follows:

(A) Compute an exponential grid around each quorum cluster. Specifically, let

$$\mathcal{I} = \bigcup_{i=1}^{m} \bigcup_{j=0}^{\lceil \log(32\xi/\varepsilon)\rceil} \mathsf{G}_{\approx}\left(\mathsf{b}(\mathsf{w}_i, 2^j \mathsf{x}_i), \frac{\varepsilon}{\zeta_1}\right) \tag{6.1}$$

be the set of grid cells covering the quorum clusters and their immediate environ, where ζ_1 is a sufficiently large constant (say, $\zeta_1 = 256\xi$).

(B) Intuitively, $\mathcal{I}$ takes care of the region of space immediately next to a quorum cluster[2]. For the other regions of space, we can apply a construction of an approximate Voronoi diagram for the centers of the clusters (the details are somewhat more involved). To this end, lift the quorum clusters into points in $\mathbb{R}^{d+1}$, as follows

$$\Sigma' = \{\Lambda'_1, \ldots, \Lambda'_m\},$$

[2] That is, intuitively, if the query point falls into one of the grid cells of $\mathcal{I}$, we can answer a query in constant time.

where $\Lambda'_i = (\mathsf{w}_i, \mathsf{x}_i) \in \mathbb{R}^{d+1}$, for $i = 1, \ldots, m$. Note that all points in Σ' belong to $U' = [0,1]^{d+1}$ by Assumption 3. Now build a $(1+\varepsilon/8)$-AVD for Σ' using the algorithm of Arya and Malamatos [2], for distances specified by the $\|\cdot\|_\oplus$ norm. The AVD construction provides a list of canonical cubes covering $[0,1]^{d+1}$ such that in the smallest cube containing the query point, the associated point of Σ', is a $(1+\varepsilon/8)$-ANN to the query point. (Note that these cubes are not necessarily disjoint. In particular, the smallest cube containing the query point q is the one that determines the assigned approximate nearest neighbor to q.)
Clip this collection of cubes to the hyperplane $x_{d+1} = 0$ (i. e., throw away cubes that do not have a face on this hyperplane). For a cube $\square$ in this collection, denote by $\mathrm{nn}'(\square)$, the point of Σ' assigned to it. Let $\mathcal{S}$ be this resulting set of canonical d-dimensional cubes.

(C) Let $\mathcal{W}$ be the space decomposition resulting from overlaying the two collection of cubes, i. e. $\mathcal{I}$ and $\mathcal{S}$. Formally, we compute a compressed quadtree $\mathcal{T}$ that has all the canonical cubes of $\mathcal{I}$ and $\mathcal{S}$ as nodes, and $\mathcal{W}$ is the resulting decomposition of space into cells. One can overlay two compressed quadtrees representing the two sets in linear time [7, 12]. Here, a cell associated with a leaf is a canonical cube, and a cell associated with a compressed node is the set difference of two canonical cubes. Each node in this compressed quadtree contains two pointers – to the smallest cube of $\mathcal{I}$, and to the smallest cube of $\mathcal{S}$, that contains it. This information can be computed by doing a BFS on the tree.
For each cell $\square \in \mathcal{W}$ we store the following.
(I) An arbitrary representative point $\square_{\mathsf{rep}} \in \square$.
(II) The point $\mathrm{nn}'(\square) \in \Sigma'$ that is associated with the smallest cell of $\mathcal{S}$ that contains this cell. We also store an arbitrary ball, $\mathbf{b}(\square) \in \mathcal{B}$, that is one of the balls completely inside the cluster specified by $\mathrm{nn}'(\square)$ – we assume we stored such a ball inside each quorum cluster, when it was computed.
(III) A number $\beta_k(\square_{\mathsf{rep}})$ that satisfies $\mathsf{d}_k(\square_{\mathsf{rep}}, \mathcal{B}) \leq \beta_k(\square_{\mathsf{rep}}) \leq (1+\varepsilon/4)\mathsf{d}_k(\square_{\mathsf{rep}}, \mathcal{B})$, and a ball $\mathrm{nn}_k(\square_{\mathsf{rep}}) \in \mathcal{B}$ that realizes this distance. In order to compute $\beta_k(\square_{\mathsf{rep}})$ and $\mathrm{nn}_k(\square_{\mathsf{rep}})$ use the data structure of Section 4, see Theorem 16.

6.3 Answering a query

Given a query point q, compute the leaf cell (equivalently the smallest cell) in $\mathcal{W}$ that contains q by performing a point-location query in the compressed quadtree $\mathcal{T}$. Let $\square$ be this cell. Let,

$$\lambda^* = \min\left(\|\mathsf{q}' - \mathrm{nn}'(\square)\|_\oplus, \beta_k(\square_{\mathsf{rep}}) + \|\mathsf{q} - \square_{\mathsf{rep}}\|\right). \tag{6.2}$$

If $\mathsf{diam}(\square) \leq (\varepsilon/8)\lambda^*$ we return $\mathrm{nn}_k(\square_{\mathsf{rep}})$ as the approximate kth-nearest neighbor, else we return $\mathbf{b}(\square)$.

6.4 Correctness

▶ **Lemma 23.** *The number* $\lambda^* = \min\left(\|\mathsf{q}' - \mathrm{nn}'(\square)\|_\oplus, \beta_k(\square_{\mathsf{rep}}) + \|\mathsf{q} - \square_{\mathsf{rep}}\|\right)$ *satisfies,* $\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \lambda^*$.

Proof. This follows by the Lipschitz property, see Observation 2. ◀

The proofs of the following lemmas appear in the full version [16].

▶ **Lemma 24.** *Let* $\square \in \mathcal{W}$ *be any cell containing* q*. If* $\mathsf{diam}(\square) \leq \varepsilon \mathsf{d}_k(\mathsf{q}, \mathcal{B})/4$*, then* $\mathrm{nn}_k(\square_{\mathsf{rep}})$ *is a valid* $(1 \pm \varepsilon)$*-approximate kth-nearest neighbor of* q*.*

▶ **Lemma 25.** *For any point* $\mathsf{q} \in \mathbb{R}^d$ *there is a quorum ball* $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ *such that (A)* Λ_i *intersects* $\mathsf{b}(\mathsf{q}, \mathsf{d}_k(\mathsf{q}, \mathcal{B}))$*, (B)* $\mathsf{x}_i \leq 3\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$*, and (C)* $\|\mathsf{q} - \mathsf{w}_i\| \leq 4\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$*.*

▶ **Definition 26.** For a given query point, any quorum cluster that satisfies the conditions of Lemma 25 is defined to be an *anchor cluster*. By Lemma 25 an anchor cluster always exists.

The following lemma, whose proof appears in the full version [16], gives a condition under which the output of the algorithm is correct.

▶ **Lemma 27.** *Suppose that among the quorum cluster balls* $\Lambda_1, \ldots, \Lambda_m$*, there is some ball* $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ *which satisfies that* $\|\mathsf{q} - \mathsf{w}_i\| \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$ *and* $\varepsilon \mathsf{d}_k(\mathsf{q}, \mathcal{B})/4 \leq \mathsf{x}_i \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$ *then the output of the algorithm is correct.*

The next lemma, whose proof appears in the full version [16], proves the correctness for the general case.

▶ **Lemma 28.** *The query algorithm always outputs a correct approximate answer, i. e., the output ball* b *satisfies* $(1 - \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$*.*

6.5 The result

The following theorem encapsulates our main result for this section. Its proof appears in the full version [16].

▶ **Theorem 29.** *Given a set* $\mathcal{B}$ *of* n *disjoint balls in* $\mathbb{R}^d$*, a number* k*, with* $1 \leq k \leq n$*, and* $\varepsilon \in (0, 1)$*, one can preprocess* $\mathcal{B}$*, in* $O\Big(n \log n + \dfrac{n}{k} C_\varepsilon \log n + \dfrac{n}{k} C'_\varepsilon\Big)$ *time, where* $C_\varepsilon = O\big(\varepsilon^{-d} \log \varepsilon^{-1}\big)$ *and* $C'_\varepsilon = O\big(\varepsilon^{-2d} \log \varepsilon^{-1}\big)$*. The space used by the data structure is* $O(C_\varepsilon n/k)$*. Given a query point* q*, this data structure outputs a ball* $b \in \mathcal{B}$ *in* $O\left(\log \dfrac{n}{k\varepsilon}\right)$ *time, such that* $(1 - \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$*.*

Note that the space decomposition generated by Theorem 29 can be interpreted as a space decomposition of complexity $O(C_\varepsilon n/k)$, where every cell has two input balls associated with it, which are the candidates to be the desired (k, ε)-ANN. That is, Theorem 29 computes a $(k.\varepsilon)$-AVD of the input balls.

7 Conclusions

In this paper, we presented a generalization of the usual $(1 \pm \varepsilon)$-approximate kth-nearest neighbor problem in $\mathbb{R}^d$, where the input are balls of arbitrary radius, while the query is a point. We first presented a data structure that takes $O(n)$ space, and the query time is $O(\log n + \varepsilon^{-d})$. Here, both k and ε could be supplied at query time. Next we presented an (k, ε)-AVD taking $O(n/k)$ space. Thus showing, surprisingly, that the problem can be solved in sublinear space if k is sufficiently large.

References

1 A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

2 S. Arya and T. Malamatos. Linear-size approximate Voronoi diagrams. In *Proc. 13th ACM-SIAM Symp. Discrete Algs.*, pages 147–155, 2002.

3 S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate spherical range counting. In *Proc. 16th ACM-SIAM Symp. Discrete Algs.*, pages 535–544, 2005.

4 S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. Assoc. Comput. Mach.*, 57(1):1–54, 2009.

5 S. Arya and D. M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17:135–152, 2000.

6 S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6):891–923, 1998.

7 M. de Berg, H. Haverkort, S. Thite, and L. Toma. Star-quadtrees and guard-quadtrees: I/O-efficient indexes for fat triangulations and low-density planar subdivisions. *Comput. Geom. Theory Appl.*, 43:493–513, July 2010.

8 P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. Assoc. Comput. Mach.*, 42:67–90, 1995.

9 P. Carmi, S. Dolev, S. Har-Peled, M. J. Katz, and M. Segal. Geographic quorum systems approximations. *Algorithmica*, 41(4):233–244, 2005.

10 K. L. Clarkson. Nearest-neighbor searching and metric space dimensions. In G. Shakhnarovich, T. Darrell, and P. Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

11 S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annual IEEE Symp. Found. Comput. Sci.*, pages 94–103, 2001.

12 S. Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. Amer. Math. Soc., 2011.

13 S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. *Theory Comput.*, 8:321–350, 2012. Special issue in honor of Rajeev Motwani.

14 S. Har-Peled and N. Kumar. Down the rabbit hole: Robust proximity search in sublinear space. In *Proc. 53rd Annual IEEE Symp. Found. Comput. Sci.*, pages 430–439, 2012.

15 S. Har-Peled and N. Kumar. Approximating minimization diagrams and generalized proximity search. In *Proc. 54th Annual IEEE Symp. Found. Comput. Sci.*, pages 717–726, 2013.

16 S. Har-Peled and N. Kumar. Robust proximity search for balls using sublinear space. *CoRR*, abs/1401.1472, 2014.

17 P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annual ACM Symp. Theory Comput.*, pages 604–613, 1998.

18 G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing).* The MIT Press, 2006.

The Benes Network is q(q−1)/2n-Almost q-set-wise Independent*

Efraim Gelman[1] **and Amnon Ta-Shma**[2]

1 **Tel-Aviv University, Tel-Aviv, Israel**
`efigelman@gmail.com.`
2 **Tel-Aviv University, Tel-Aviv, Israel**
`amnon@tau.ac.il`

Abstract

A switching network of depth d is a layered graph with d layers and n vertices in each layer. The edges of the switching network do not cross between layers and in each layer the edges form a partial matching. A switching network defines a stochastic process over $\mathbb{S}_n$ that starts with the identity permutation and goes through the layers of the network from first to last, where for each layer and each pair (i, j) in the partial matching of the layer, it applies the transposition (ij) with probability half. A switching network is *good* if the final distribution is close to the uniform distribution over $\mathbb{S}_n$.

A switching network is ε-almost q-permutation-wise independent if its action on any ordered set of size q is almost uniform, and is ε-almost q-set-wise independent if its action on any set of size q is almost uniform. Mixing of switching networks (even for q-permutation-wise and q-set-wise independence) has found several applications, mostly in cryptography. Some applications further require some additional properties from the network, e. g., the existence of an algorithm that given a permutation can set the switches such that the network generates the given permutation, a property that the Benes network has.

Morris, Rogaway and Stegers showed the Thorp shuffle (which corresponds to applying two or more butterflies one after the other) is q-permutation-wise independent, for $q = n^\gamma$ for γ that depends on the number of sequential applications of the butterfly network. The techniques applied by Morris et al. do not seem to apply for the Benes network.

In this work we show the Benes network is almost q-set-wise independent for q up to about $\sqrt{n}$. Our technique is simple and completely new, and we believe carries hope for getting even better results in the future.

1998 ACM Subject Classification C.2 Computer-communication Networks

Keywords and phrases switching network, mixing, Benes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.327

1 Introduction

The subject of this paper is the generation of a random permutation through the composition of (few) simple building blocks. The question is quite old and appears in many different contexts, one prominent example is the application to CCA-security in cryptography. Generating an almost random permutation on $\log n$-bit strings in such a way, ensures CCA-security to almost 2^n queries (see for example [14]). The problem comes in two main flavors:

* Supported by the Israel science Foundation (grants no. 1090/10 and 994/14) and by the United States-Israel Binational Science Foundation grant no. 2010120.

Mixing with simple permutations: Here, we have a small subset of (usually simple) permutations, and we ask how fast the process of composing random elements from the subset converges to the uniform distribution over $\mathbb{S}_n$. Some typical examples are, e. g., when:

- The subset that contains all transpositions,
- The subset that contains all permutations whose cycle decomposition is $2^{n/2}$, i. .e., they are a product of $n/2$ disjoint transpositions, and,
- The subset that contains all permutations in some conjugacy class of $\mathbb{S}_n$.

A seminal result of Diaconis and Shahshahani [6] analyzes the case where the conjugacy class is the set of all transpositions, and shows that the mixing time is about $\frac{1}{2}n \log n + \Theta(n)$ steps. This result was later extended to the conjugacy class of r-cycles [11]. The technique uses non-commutative Fourier transform and combinatorics of Young Tableaux, and has been used with great success in many works.

Switching networks: A switching network of depth d is a layered graph with d layers and n vertices in each layer. The edges of the switching network do not cross between layers and in each layer the edges form a partial matching. A switching network defines a stochastic process over $\mathbb{S}_n$ that starts with the identity permutation and goes through the layers of the network from first to last, where for each layer and each pair (i, j) in the partial matching of the layer, it applies the transposition (ij) with probability half. A switching network is *good* if the final distribution is close to the uniform distribution over $\mathbb{S}_n$.

One may view the switching network problem as a *derandomized* version of mixing with simple permutations. For example, Diaconis and Shahshahani showed that composing $d = \frac{1}{2}n \log n + \Theta(n)$ *random* transpositions is sufficient and necessary for getting a (close to) uniform permutation. However, a switching network would show a specific sequence $\sigma_1, \ldots, \sigma_t$, such that picking $i_1, \ldots, i_t \in \{0, 1\}$ at random and applying $\sigma_t^{i_t} \ldots \sigma_1^{i_1}$ would generate a close to uniform distribution over $\mathbb{S}_n$.

Similarly, the *matching-exchange* process goes as follows. A random perfect matching on the n elements is chosen uniformly at random, and for each pair (i, j) in the matching, the transposition (i, j) is applied with probability half. One way to analyze this process is by choosing k (the number of transpositions) between 0 and $n/2$ according to the binomial distribution, and then choosing a random permutation from the conjugacy class $2^k 1^{n-2k}$. Using non-commutative Fourier analysis and character estimates [12] proved the process converges to uniform in $O(\log n)$ time. Another proof of the same result, but with inferior constants, was obtained using delayed path coupling [4].

Building a switching network would show a specific sequence of perfect matchings that can be used in the matching-exchange process. Thus, constructing a switching network essentially amounts to de-randomizing the matching-exchange process, thus limiting the use of randomness to the bare minimum needed for generating a distribution close to uniform over $\mathbb{S}_n$.

Moreover, it turns out in some important applications (e. g., CCA security) it is crucial to have the second variant (of a mixing network) rather than the first one (of mixing with simple permutations). Another application is for the security of electronic voting and the efficient zero knowledge proofs for the decryption of the encrypted votes (see for example: [2, 1] and [10]). In this application both variants may be used.

However, the question whether good shallow switching networks exist is still wide open. In 1981, David Chaum [3], suggested a cryptographic protocol that guarantees anonymous communication, provided that shallow switching networks exist. In 1993, Rackoff and Simon [15] claimed an explicit switching network of depth $poly(\log n)$ is good. Rackoff and Simon gave only a short sketch of the proof while the full proof of the theorem, which had been delayed to the journal version of the paper, was never published. In 1999, Czumaj et al. [4] claimed the existence of a good switching network of depth $O(\log^2 n)$, however, the correctness proof was deferred to the full version of the paper and has not appeared to date. Finally, it has been proven in [5] using delayed path coupling that:

▶ **Theorem 1** ([5]). *There exists an explicit construction of a good switching network of depth polylog*(n).[1]

Morris [13] constructed, using totally different techniques, an explicit switching network of depth $O(\log^4 n)$ that builds upon Thorp shuffle. We expand on this shortly. Still, in our current state of knowledge, it is not known whether good switching networks of *logarithmic* depth exist.

1.1 q-wise independence

One way to visualize the action of a switching network is as follows. Assume we have a switching network with n vertices and d layers. Put n numbered balls on the n vertices of the first layer. Go through the layers from first to last, and for each layer, and each pair (i, j) in the partial matching of the layer, with probability half switch the balls that are currently at positions i and j. A network is good if the distribution of the n numbered balls at the last layer is close to uniform.

One may weaken this definition by considering the action of the network on only $q \leqslant n$ numbered balls. Further weakening is considering the case where the balls are *identical* balls.

- We say a network is ε-almost *q-permutation-wise* independent if for any possible way of putting q numbered balls, the distribution of the balls at the last layer is ε-close to the uniform distribution on all the $\frac{n!}{(n-q)!}$ possible end configurations.
- We say a network is ε-almost *q-set-wise* independent if for any possible way of putting q identical balls, the distribution of the balls at the last layer is ε-close to the uniform distribution on all the $\binom{n}{q}$ possible end configurations.

A network that is q permutation–wise independent is in particular q set–wise independent.

Mathematically, the q-wise independence problem is natural. The *permutation-wise* problem compares the action of the network and the action of $\mathbb{S}_n$ on the set $X = [n]_q$ of all *ordered* subsets of size q, and the *set-wise* problem does the same for the set $X = \binom{[n]}{q}$ of all subsets of size q. The q-wise question was extensively studied, e. g., by Gowers [7] and later by Hoory et al. [8], who show, using canonical paths, that composing few random simple permutations from a certain fixed family is ε-almost q-wise. Morris, Rogaway and Stegers [14] showed the Thorp shuffle, that we soon discuss, is almost q permutation–wise independent for a large q, and we soon expand on their result.

1.2 The Thorp shuffle and the butterfly network

We first introduce some notation. For $0 \leq x < n$ let $x = x_{\log n}, \ldots, x_1$ be its binary representation. Let $\underline{x}$ (resp. $\overline{x}$) be the integer that has the same binary representation as x except that its most significant bit is 0 (resp. 1) independent of the most significant bit of x.

[1] It seems that the constant in the exponent is currently at least 7.

I. e.,

$$\begin{aligned} \underline{x} &= 0, x_{\log n-1}, \dots, x_1 \\ \overline{x} &= 1, x_{\log n-1}, \dots, x_1 \end{aligned}$$

For example, if $n = 16$, $x = 11$ then $\underline{x} = 3$ and $\overline{x} = 11$.

Now we introduce the Thorp shuffle. At each stage of the Thorp shuffle we do the following:

- For all $0 \leq x < n/2$, with probability half we switch the elements in cells number $\underline{x}$ and $\overline{x}$ (and keep them in place with probability half).
- We permute the elements as follows: an element that is at cell number $x = x_{\log n}, \dots, x_1$ is moved to cell number $x_{\log n-1}, \dots, x_1, x_{\log n}$. Notice that this forms a permutation over the n elements.

This process is equivalent to the butterfly switching network. In the butterfly switching network we have $\log n$ layers with n vertices in each layer. We index the n elements by their binary representation in $\{0,1\}^{\log n}$. The edges of the i'th layer, for $i = 1, \dots, \log n$, connect $x, y \in \{0,1\}^{\log n}$ if and only if they differ only in the i'th coordinate. The Thorp shuffle for $T = r \log n$ stages is equivalent to applying r butterfly switching networks one after the other.

One way to visualize what the butterfly switching network does is as follows. Take the $(\log n)$ dimensional cube with its usual edges, i. .e. $x, y \in \{0,1\}^{\log n}$ are neighbors if and only if they differ only in one coordinate. For every i and edge $(x, x + e_i)$ color the edge with color i. As before, put n numbered balls on the n vertices of the cube, go sequentially over $i = 1, \dots, \log n$, and for each edge (x, y) colored i, switch the balls on x, y with probability half.

The butterfly switching network (or equivalently the thorp shuffle with $\log n$ stages) induces a probability distribution over permutations of $\mathbb{S}_n$. What can we say about it?

Clearly, the butterfly network is 1-wise (set and permutation) *perfect*, i. .e. if we put *one* ball at any starting vertex $x \in \{0,1\}^{\log n}$ and apply the network, the ball will have the same $(\frac{1}{n})$ probability to finish at any vertex. However, if we look at two balls, things are not that good. Suppose we start with two balls on vertices x_1, x_2 that are connected by an edge colored 1, i. .e., $x_2 = x_1 + e_1$. Notice that after stage 1 the two balls have different first coordinate, and this does not change later on. Thus, the two balls must end up at vertices that differ in their first bit. A random permutation, however, would do that only with probability half. Thus, the induced distribution over $\mathbb{S}_n$ is far from uniform (set and permutation wise).

In [14] , Morris et al. analyze the Thorp shuffle. They show using a clever coupling argument, that applying the Thorp shuffle $T = 2r \log n$ stages, is $\frac{q}{r+1} \cdot (\frac{4q \log n}{n})^r$-almost q-wise-permutation independent. For example running the butterfly twice (i. .e., $r = 1$) we get $\varepsilon = \frac{2q^2 \log n}{n}$ and we may have q almost as large as $\sqrt{n}$.

1.3 The Benes network

Another classical switching network is the Benes network. The Benes network is applying the butterfly twice, first in its original order, and then in reversed order. Thus, the Benes network is similar to the Thorp shuffle in that it repeatedly uses the butterfly network, but it differs from it in the way it orders the layers. An example of the Benes network on eight vertices is given in Figure 1.

It is well known that the Benes network gives positive probability to each permutation $\pi \in \mathbb{S}_n$, see., e.g., [9, Section 3.2]. Other than that, not much was known about the distribution induced over $\mathbb{S}_n$ by the Benes network. The techniques of Morris et al. do not seem to work for the Benes network.[2]

The Benes network is mathematically very natural (as it makes the operator Hermitian, see Sec 2.1). It also induces a clean recursive structure, i. .e., a Benes network on n elements is composed of simple first and last layers, and two parallel Benes networks on $n/2$ elements, see Section 2.1 and Figure 1. This clean recursive structure is also behind the proof that every permutation may be obtained by the Benes network.

Also, the Benes network appears in many applications, and often the reason is exactly this clean structure it possesses. For example, Abe [2] uses the Benes network for mixnets (and electronic voting) and a key property that is required is the ability to easily route any permutation on the network, a property that, as far as we know, the thorp shuffle lacks.

There are therefore two main reasons to study the Benes network: the first is that it appears in protocols that require some specific property of it. The second is that it is mathematically elegant, and the hope that one may be able to use its elegant recursive structure to finally give a construction of a good logarithmic-depth switching network. In particular, it is possible that applying sequentially a constant number of Benes networks, defines a distribution that is close to uniform over $\mathbb{S}_n$.

In this work we use the elegant structure of the network and prove that the Benes network is ε-almost q-set-wise independent, for q up to about $\sqrt{n}$. Specifically, we prove:

▶ **Theorem 2.** *The Benes network is $\frac{q(q-1)}{2n}$-almost q-set-wise independent.*

Parameter-wise, the result we obtain is slightly better than the one obtained in [14] for the corresponding Thorp shuffle (with $2\log n$ stages). On the downside, [14] show q-**permutation**-wise independent and also show that applying a series of Thorp shuffle sequentially significantly reduces the error, while we do not show the corresponding fact for the Benes network.

More importantly, we believe the proof technique, that is completely different than the one in [14], is of independent interest, and reveals the delicate and beautiful structure of the Benes network. We give a recursive formula for the probability the Benes moves a given set of q elements to another, and we identify the crucial parameters on which it depends. The question then reduces to a combinatorial question, that can be solved by analyzing two related experiments. Using this we prove that any set of q elements is obtained with probability at least $\frac{q!}{n^q}$ (and notice how close this is to the uniform probability of $\frac{1}{\binom{n}{q}}$) which is a strong and surprising result by itself. We believe there might be a way to tighten the analysis given in the paper and achieve much better results (e. g., proving q above $\sqrt{n}$). We view this work as a first step of understanding the Benes switching network and believe it sheds light on the way it operates and we hope it may possibly lead to solving the challenging problem of constructing a good switching network of logarithmic depth.

2 Definitions and Notation

For a set S, $\binom{S}{q}$ is the set of all q-subsets of S, i. .e., $\{A \subseteq S \mid |A| = q\}$. We let $[n]$ denote the set $\{0, \ldots, n-1\}$. The symmetric group $\mathbb{S}_n$ acts on $\binom{[n]}{q}$ in a natural way, $\pi(A) =$

[2] One key ingredient in the proof is the fact that using the update rule described in the paper, for time $t \geqslant \log n - 1$, the probability that any two elements are adjacent at time t is $\leqslant 2^{1-\log n}$. This does not hold for the Benes network since with this update rule the probability can only be bounded by $\leqslant 2^{1-0.5\log n}$.

$\{\pi(a) \mid a \in A\}$. Clearly the action is transitive.

Suppose $\mathbb{S}_n$ acts on a set X. We define π_X to be the $|X| \times |X|$ matrix, where $(\pi_X)_{i,j} = \delta_{\pi(x_j),x_i}$. We specify two special cases:

- When $X = \mathbb{S}_n$ and the action is by left multiplication, we denote the matrix π_X by π.
- When $X = \binom{[n]}{q}$ and the action is the natural action defined above, we denote the matrix π_X by π_q.

For a distribution $\mathfrak{D}$ over $\mathbb{S}_n$ we let D_X denote the $|X| \times |X|$ matrix

$$D_X \quad = \quad \sum_{\pi \in \mathbb{S}_n} \mathfrak{D}(\pi) \cdot \pi_X. \tag{1}$$

If H is a subset of $\mathbb{S}_n$, we identify it with the flat distribution over H, and let H_X be the corresponding matrix.

Notice that $(D_X)_{i,j} = \Pr_{\pi \in \mathfrak{D}} [\ \pi(x_j) = x_i\ ]$, i. .e., (D_X) describes the transition matrix of the stochastic process that picks π according to the distribution $\mathfrak{D}$ and applies it on x (or a distribution over X).

2.1 The Benes network

Given $x, y \in [n]$, we say $x \overset{i}{\sim} y$ if their binary representation differs only on bit number i (starting with the least significant bit). For example $13 \overset{2}{\sim} 15$ since their binary representations are 1101 and 1111.

A Benes network is a layered graph with $2\log n$ layers, indexed from 1 to $2\log n$ and n vertices in each layer. In layers i and $2\log n + 1 - i$ (for $i = 1, \ldots, \log n$), the matching is formed by all the edges (x, y) s.t. $x \overset{\log n+1-i}{\sim} y$.

Alternatively, let H^i (for $i = 1, 2, \ldots, \log n$) be the abelian subgroup of $\mathbb{S}_n$ generated by the set of $n/2$ disjoint transpositions

$$\left\{ (x \quad y) \mid x \overset{i}{\sim} y \right\}.$$

A Benes network is a series $H^{\log n}, \ldots, H^1, H^1, \ldots, H^{\log n}$ indexed by n.

A Benes network defines a distribution $\mathfrak{B}^{(n)}$ on $\mathbb{S}_n$ as follows: pick $\pi_{\log n} \in H^{\log n}, \ldots, \pi_1 \in H^1, \sigma_1 \in H^1, \ldots, \sigma_{\log n} \in H^{\log n}$ and output $\sigma_{\log n} \cdot \ldots \sigma_1 \cdot \pi_1 \cdot \ldots \pi_{\log n} \in \mathbb{S}_n$. We denote

$$B_X^{(n)} \quad = \quad \Sigma_{\pi \in \mathbb{S}_n} \mathfrak{B}^{(n)}(\pi) \cdot \pi_X. \tag{2}$$

As before, when $X = \mathbb{S}_n$ and the action is by left multiplication, we denote $B_X^{(n)}$ simply by $B^{(n)}$, and if $X = \binom{[n]}{q}$ and the action is the natural action, we denote $B_X^{(n)}$ by $B_q^{(n)}$. When n is clear from the context we omit it and write $\mathfrak{B}, B_X, B_q$ and B instead of $\mathfrak{B}^{(n)}, B_X^{(n)}, B_q^{(n)}$ and $B^{(n)}$ respectively. Notice that

$$B_X \quad = \quad H_X^{\log n} \cdot \ldots \cdot H_X^1 \cdot H_X^1 \ldots H_X^{\log n}. \tag{3}$$

Looking at Eq (3) we see that for every X, B_X is Hermitian and positive. Looking at Eq (2) we see that B_X is stochastic. In particular B_X is also doubly stochastic.

As B_X is Hermitian we have that $(B_X^n)_{\alpha,\beta} = (B_X^n)_{\beta,\alpha}$ for any $\alpha, \beta \in X$. Since for any $\pi \in \mathbb{S}_n$ and $A \subseteq [n]$ we have that $\pi(A^c) = (\pi(A))^c$, then $(B_q^n)_{\alpha,\beta} = (B_{n-q}^n)_{\alpha^c,\beta^c}$ for any $A_1, A_2 \in [n]$, where c denotes set complement, i. .e., $A^c = [n] - A$.

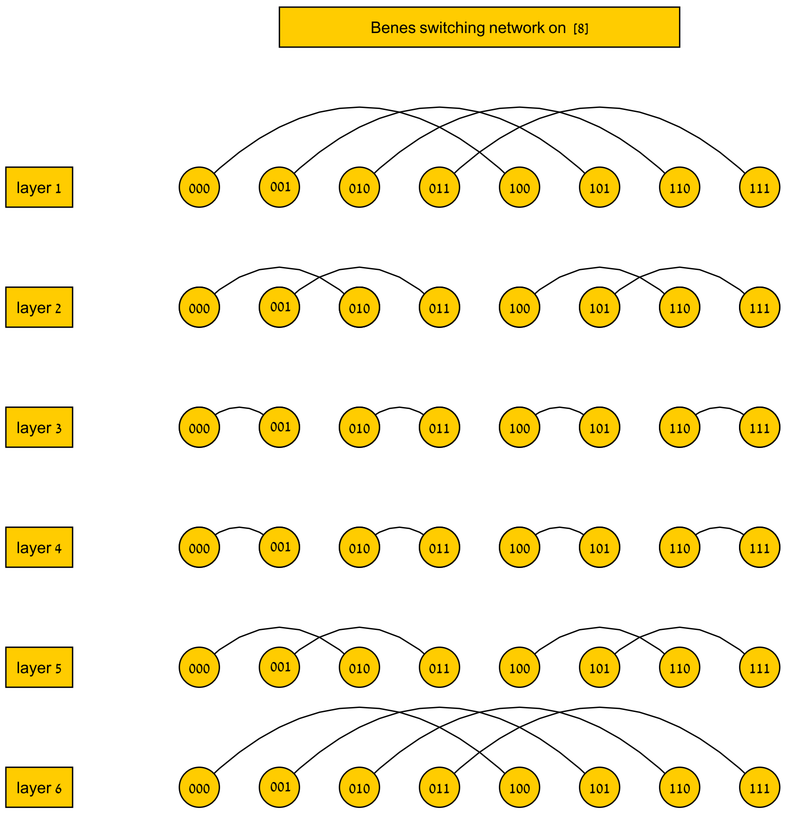


Figure 1 The Benes switching network on eight vertices. The nodes are the elements of $\{0, \dots, 7\}$ in binary representation. Notice that if we look at layers 2-5 and omit the most significant bit, we get the Benes network on four vertices at the left half and right half of the network.

2.2 Almost q-set-wise independence

▶ **Definition 3** (Almost q-set-wise Independence)**.** Let $\mathfrak{D}$ be a distribution over $\mathbb{S}_n$, and q an integer. We say $\mathfrak{D}$ is ε–almost q–set-wise independent in norm ℓ, if for any initial distribution v_0 over $X = \binom{[n]}{q}$,

$$\|D_X v_0 - \mathfrak{U}_X\|_\ell \le \varepsilon,$$

where D_X is the $|X| \times |X|$ matrix defined in Eq (1) and $\mathfrak{U}_X$ is the uniform distribution over X.

As usual, it is enough to show that

$$\|D_X v_0 - \mathfrak{U}_X\|_\ell \le \varepsilon,$$

for initial distributions v_0 that are 1 on one element of X and zero otherwise and use the convexity of the norm.

It is well known that $\mathfrak{B}$ is perfectly one-wise independent, i.e

$$B_1 = \frac{1}{n} J, \qquad (4)$$

where J is the all-one matrix.

3 Benes is $\frac{q(q-1)}{2n}$-almost q-set-wise independent

▶ **Theorem 4.** *For every q and n such that $\binom{q}{2} \leqslant n$, the Benes network is $\frac{q(q-1)}{2n}$-almost q-set-wise independent*

Proof. Fix $X = \binom{[n]}{q}$. Take $\beta \in \binom{[n]}{q}$ and let v_0 be the distribution that is 1 on β. By our previous remark , it is enough to show that:

$$\|(B_q^n)v_0 - \mathfrak{U}_X\|_1 \overset{def}{\geqslant} \sum_{\alpha\in X\colon\ (B_q^n)_{\alpha,\beta}<U_q} (U_q - (B_q^n)_{\alpha,\beta}) \leq \varepsilon,$$

where $U_q = \frac{1}{\binom{n}{q}}$. The central ingredient in the proof is showing that:

▶ **Lemma 5** (Main). *For every $q \leq n$ where n is a power of 2 and every $\alpha, \beta \in \binom{[n]}{q}$ we have that $(B_q^n)_{\alpha,\beta} \geqslant \frac{q!}{n^q}$*

Having the lemma we see that

$$\begin{aligned}
\sum_{\alpha\colon\ (B_q^n)_{\alpha,\beta}<U_q} (U_q - (B_q^n)_{\alpha,\beta}) &< \sum_{\alpha\in\binom{[n]}{q}} (U_q - \frac{q!}{n^q}) \\
&= \sum_{\alpha\in\binom{[n]}{q}} U_q - \sum_{\alpha\in\binom{[n]}{q}} \frac{q!}{n^q} \\
&= 1 - \binom{n}{q}\frac{q!}{n^q} \\
&= 1 - (1-\frac{1}{n})(1-\frac{2}{n})\dots(1-\frac{q-1}{n}) \\
&< \sum_{i=0}^{q-1} \frac{i}{n} = \frac{q(q-1)}{2n}.
\end{aligned}$$

Whereas the first inequality is using the lemma and summing over all $\alpha \in \binom{[n]}{q}$, the equalities are simple algebra and the second inequality is simply inclusion-exclusion principle: Think of independent events $A_i, i = 1 \dots m$,with probabilities p_i. Then the probability that at least one of the event occurs is given by $1 - (1 - p_1)(1 - p_2) \dots (1 - p_m)$ and is smaller then $\sum_{i=1}^{m} p_i$. Now take $m = q - 1$ and $p_i = \frac{i}{n}$ and the inequality follows. ◀

Before proving the main lemma we will look into the recursive structure of the Benes network and obtain a recursive formula for $(B_q^n)_{\alpha,\beta}$.

4 The recursive structure of the Benes network

Looking at the Benes network, we notice that if we take the series

$$\underline{H}^{\log n-1}, \dots, \underline{H}^1, \underline{H}^1, \dots, \underline{H}^{\log n-1}$$

where $\underline{H}^i$ (for $i = 1, \dots, \log n - 1$) is the subgroup of $\mathbb{S}_n$ generated by the set of $n/4$ transpositions $\left\{(x \quad y) \mid x \overset{i}{\sim} y \text{ ,} x < n/2\right\}$, we get a Benes network on $[n/2]$.

Taking $\overline{H}^{\log n-1}, \dots, \overline{H}^1, \overline{H}^1, \dots, \overline{H}^{\log n-1}$, where $\overline{H}^i$ (for $i = 1, \dots, \log n - 1$) is the subgroup of $\mathbb{S}_n$ generated by the set of $n/4$ transpositions $\left\{(x \quad y) \mid x \overset{i}{\sim} y \text{ ,} x \geqslant n/2\right\}$, gives us a switching network on $\{n/2, \dots, n-1\}$ that is isomorphic to the Benes network on $[n/2]$.

We now want to use this recursive structure to get a recursive formula for $(B^n_q)_{\alpha,\beta}$, which is the probability that if we choose $\pi_{\log n} \in H^{\log n}, \ldots, \pi_1 \in H^1, \sigma_1 \in H^1, \ldots, \sigma_{\log n} \in H^{\log n}$ (all with flat probability $\frac{1}{2^{\frac{n}{2}}}$) then $\sigma_{\log n} \cdot \ldots \sigma_1 \cdot \pi_1 \cdot \ldots \pi_{\log n}(\beta) = \alpha$.

The first and last layers of the Benes network connect two inputs that differ only in the most significant bit. We use the notation of $\underline{x}$ and $\overline{x}$ introduced earlier. We remind the reader that $\underline{x} \overset{\log n}{\sim} \overline{x}$ and that either $x = \underline{x}$ or $x = \overline{x}$

Given $\alpha \subseteq [n]$, We denote

$$\begin{aligned} \underline{\alpha} &= \{\underline{x} | x \in \alpha\} \\ \overline{\alpha} &= \{\overline{x} | x \in \alpha\} \end{aligned}$$

We say $\beta \in X = \binom{[n]}{q}$ contains a pair $(\underline{x}, \overline{x})$ if both $\underline{x}, \overline{x}$ belong to β. We observe that if $\pi \in H^{\log n}$ then $\pi\{\underline{x}, \overline{x}\} = \{\underline{x}, \overline{x}\}$. An element $x \in [n]$ is called *paired* in β if β contains $(\underline{x}, \overline{x})$.

▶ **Proposition 1** (Recursive Formula). *Fix α and β in $X = \binom{[n]}{q}$. Assume now β contains r pairs $(\underline{x}, \overline{x})$ and α contains d such pairs. Denote R and D the set of* paired *elements in β and α respectively. Either r or d may be zero. Denote by R_1 the set of* paired *elements in β that are on the left side of the network (i. .e. belong to $[n/2]$). Symmetrically denote by R_2 the set of* paired *elements in β that are on the right side of the network (i. .e. belong to $\{n/2, \ldots, n-1\}$). Notice that $R_2 = \overline{R_1}$ and that $R = R_1 \bigcup R_2$. We denote respectively the sets D_1, D_2 for α. So $|R_1| = |R_2| = r$ and $|D_1| = |D_2| = d$. In this setting we have:*

$$(B^n_q)_{\alpha,\beta} \quad = \quad \frac{1}{2^{q-2r}} \cdot \frac{1}{2^{q-2d}} \sum_{\beta_L : R_1 \subseteq \beta_L \subseteq \beta - R_2} \; \sum_{\alpha_L : D_1 \subseteq \alpha_L \subseteq \alpha - D_2} (B^{\frac{n}{2}}_{|\beta_L|})_{\underline{\alpha}_L, \underline{\beta}_L} (B^{\frac{n}{2}}_{q-|\beta_L|})_{\overline{\alpha - \underline{\alpha}_L}, \overline{\beta - \underline{\beta}_L}}$$

Proof of Proposition 1. The action of the first layer of the Benes network on β is captured by the set $\beta_L \subseteq \beta$ of elements of β that are moved by the first layer to the left side. Clearly, for any $\pi_{\log n} \in H^{\log n}$ we have that $\pi_{\log n}(R) = R$. Thus, paired elements in β are left in place, so R_1 stays on the left side and R_2 on the right side. Non-paired elements, i. .e., the set $\beta - R$ can be either moved to the left or to the right, and each possibility occurs with equal probability.

Thus, to transfer β to α through the Benes network we have to go over all the possibilities to choose β_L s.t. $R_1 \subseteq \beta_L \subseteq \beta - R_2$. Once we choose β_L, we have to route $\underline{\beta_L}$ through the left Benes network to some $\underline{\alpha_L}$ and $\overline{\beta - \beta_L}$ through the right Benes network to $\overline{\alpha - \underline{\alpha_L}}$. The last layer then routes $\underline{\alpha_L} \cup \overline{\alpha - \underline{\alpha_L}}$ to α.

Let us see what paths are possible. First, as we saw before, we have freedom to choose any $R_1 \subseteq \beta_L \subseteq \beta - R_2$. Next, we have freedom to choose any α_L such that $|\alpha_L| = |\beta_L|$ and $D_1 \subseteq \alpha_L \subseteq \alpha - D_2$. The probability of such an event is $\frac{1}{2^{q-2r}}$ for the probability the first layer is captured by β_L, $\frac{1}{2^{q-2d}}$ for the probability the last layer is captured by α_L, and for each α_L, β_L as above, $(B^{\frac{n}{2}}_{|\beta_L|})_{\underline{\alpha}_L, \underline{\beta}_L}$ for the probability the left Benes network routes $\underline{\beta}_L$ to $\underline{\alpha}_L$, and $(B^{\frac{n}{2}}_{q-|\beta_L|})_{\overline{\alpha - \underline{\alpha}_L}, \overline{\beta - \underline{\beta}_L}}$ for the probability the right Benes network routes $\overline{\beta - \underline{\beta}_L}$ to $\overline{\alpha - \underline{\alpha}_L}$. ◀

We now prove the main lemma.

5 Proof of main lemma

Proof of Lemma 5. We prove the lemma by induction on q using the recursive structure of the Benes network. The base case is $q = 1$ and arbitrary n and is very simple: $(B^n_1)_{\alpha,\beta} = \frac{1}{n}$ for every $\alpha, \beta \in \binom{[n]}{1}$. We now assume for all $q \leq q_0$ and we prove for $q = q_0 + 1$.

We prove the case $q = q_0 + 1$ by induction on n. The base case is for $n = n_q$ where n_q is the first power of 2 such that $q \leqslant n_q$, i. .e. $\frac{n_q}{2} < q \leqslant n_q$. In this case

$$(B_q^{n_q})_{\alpha,\beta} \quad = \quad (B_{n_q-q}^{n_q})_{\alpha^c,\beta^c} \geqslant \frac{(n_q-q)!}{n_q^{n_q-q}} \geqslant \frac{q!}{n_q^q},$$

where we have used the simple fact that $n_q - q \le q_0$ and the induction hypothesis on q.

Now, fix α and β in $X = \binom{[n]}{q}$ with the same setting as proposition 1. We get:

$$\begin{aligned}
(B_q^n)_{\alpha,\beta} &= \frac{1}{2^{q-2r}} \cdot \frac{1}{2^{q-2d}} \sum_{\beta_L: R_1 \subseteq \beta_L \subseteq \beta - R_2} \sum_{\alpha_L: D_1 \subseteq \alpha_L \subseteq \alpha - D_2} (B_{|\beta_L|}^{\frac{n}{2}})_{\underline{\alpha}_L, \underline{\beta}_L} (B_{q-|\beta_L|}^{\frac{n}{2}})_{\overline{\alpha - \underline{\alpha}_L}, \overline{\beta - \underline{\beta}_L}} \\
&= \frac{1}{2^{q-2r}} \cdot \frac{1}{2^{q-2d}} \sum_{i=r}^{q-r} \sum_{\beta_L: R_1 \subseteq \beta_L \in \binom{\beta - R_2}{i}} \sum_{\alpha_L: D_1 \subseteq \alpha_L \in \binom{\alpha - D_2}{i}} (B_i^{\frac{n}{2}})_{\underline{\alpha}_L, \underline{\beta}_L} (B_{q-i}^{\frac{n}{2}})_{\overline{\alpha - \underline{\alpha}_L}, \overline{\beta - \underline{\beta}_L}} \\
&\geqslant \frac{1}{2^{q-2r}} \cdot \frac{1}{2^{q-2d}} \sum_{i=r}^{q-r} \binom{q-2r}{i-r} \binom{q-2d}{i-d} \frac{i!}{(\frac{n}{2})^i} \frac{(q-i)!}{(\frac{n}{2})^{q-i}} \\
&= \frac{(q-2d)!}{n^q} \frac{(d!)^2}{2^{q-2r-2d}} \cdot \sum_{i=r}^{q-r} \binom{q-2r}{i-r} \binom{i}{d} \binom{q-i}{d}
\end{aligned}$$

where the inequality is by induction on n and the final equality is simple algebra.

We want to show that $\frac{(q-2d)!}{n^q} \frac{(d!)^2}{2^{q-2r-2d}} \cdot \sum_{i=r}^{q-r} \binom{q-2r}{i-r} \binom{i}{d} \binom{q-i}{d} \geqslant \frac{q!}{n^q}$. This follows from our main combinatorial lemma:

▶ **Lemma 6** (Combinatorial Lemma). *Let $d \leqslant r$ then:*

$$(q-2d)! \frac{(d!)^2}{2^{q-2r-2d}} \cdot \sum_{i=r}^{q-r} \binom{q-2r}{i-r} \binom{i}{d} \binom{q-i}{d} \geqslant q!$$

◀

We now prove the combinatorial lemma.

Proof of Lemma 6. We denote $(N)_l = N(N-1) \cdot \ldots (N-l+1) = \frac{N!}{(N-l)!}$, so if $l > N$ then $(N)_l = 0$. The inequality is equivalent to

$$\sum_{i=r}^{q-r} \frac{\binom{q-2r}{i-r}}{2^{q-2r}} (i)_d (q-i)_d \geqslant \frac{(q)_{2d}}{2^{2d}} \,. \tag{5}$$

We consider two experiments. In both experiments we have q numbered balls, and two colors black and white. In the first experiment we color all the q balls uniformly at random. In the second experiment r of the balls are already colored white and r are colored black and we color the remaining $q - 2r$ balls uniformly at random. In both experiments we define a random variable X whose value is the number of possibilities to choose an ordered sequence of d white balls and an ordered sequence of d black balls. I. e., if C is the random variable counting the number of balls that are colored white, then $X = (i)_d (q-i)_d$ given that $C = i$. Now we compute the expected value of X in each of the two experiments.

In the first experiment $E(X) = \sum_{i=d}^{q-d} \frac{\binom{q}{i}}{2^q} (i)_d (q-i)_d$, and rearranging:

$$\begin{aligned}
\sum_{i=d}^{q-d} \frac{\binom{q}{i}}{2^q} (i)_d (q-i)_d &= \frac{1}{2^q} \sum_{i=d}^{q-d} \frac{q!}{i!(q-i)!} (i)_d (q-i)_d \\
&= \frac{1}{2^q} \sum_{i=d}^{q-d} \frac{q!}{(i-d)!(q-i-d)!} = \frac{(q)_{2d}}{2^q} \sum_{i=d}^{q-d} \frac{(q-2d)!}{(i-d)!(q-i-d)!} \\
&= \frac{(q)_{2d}}{2^q} \sum_{i=d}^{q-d} \binom{q-2d}{i-d} = \frac{(q)_{2d}}{2^q} 2^{q-2d} = \frac{(q)_{2d}}{2^{2d}} .
\end{aligned}$$

Thus, this is exactly the right term in inequality (5). Also, in the second experiment $E(X) = \sum_{i=r}^{q-r} \frac{\binom{q-2r}{i-r}}{2^{q-2r}} (i)_d (q-i)_d$, which is the left term in the inequality. Thus, we need to prove that $E(X)$ in the second experiment is larger than $E(X)$ in the first experiment.

Let t_i (p_i) denote the probability that $C = i$ in the first (second) experiment respectively. Namely, $t_i = \frac{\binom{q}{i}}{2^q}$ and $p_i = \frac{\binom{q-2r}{i-r}}{2^{q-2r}}$. Let $X_i = (X|C = i) = (i)_d (q-i)_d$. It is easy to see that:

▶ **Claim 1.** *If i, j are integers, $(d \leqslant i < j \leqslant \frac{q}{2})$, then $t_i < t_j$ and $p_i < p_j$.*

Also:

▶ **Lemma 7.** *If i, j are integers, $(d \leqslant i < j \leqslant \frac{q}{2})$ then $X_i < X_j$.*

▶ **Lemma 8.** *If $d \leqslant i < j \leqslant \frac{q}{2}$ then $\frac{p_j}{p_i} > \frac{t_j}{t_i}$.*

Both lemmas are proven easily by induction. Take $j = i + 1$, and by simple algebra one can see that the inequalities in both lemmas are equivalent to $2i < q - 1$ which is true since i is an integer and $i < \frac{q}{2}$. For example, $\frac{p_i}{p_{i+1}} = \frac{i+1-r}{q-r-i}$ and therefore $p_i < p_{i+1}$ iff $i + 1 - r < q - r - i$ iff $2i < q - 1$.

Now, from these two lemmas together with the symmetry around $\frac{q}{2}$ of our variables and probabilities ($X_j = X_{q-j}, t_j = t_{q-j}, p_j = p_{q-j}$), we get that there exists some index $i_0 \leqslant \frac{q}{2}$ such that $p_i \geqslant t_i$ if and only if $i \in S = \{i_0, i_0 + 1 \ldots, q - i_0\}$. We also know that since $\{X_i\}, i = 1, \ldots, \frac{q}{2}$ is monotonically increasing, it must be that for any $i \in S, j \in S^c$ we have that $X_i \geqslant X_j$. We now prove this implies that the expected value of the second experiment is larger then that of the first experiment:

▶ **Lemma 9.** *Let $\{X_i\}_{i \in A}$ be a set of non negative variables. Assume $S \subset A$ such that:*

- *For any $i \in S, j \in S^c$ we have that $X_i \geqslant X_j$, i.e $min_{i \in S}\{X_i\} \geqslant max_{i \in S^c}\{X_i\} = X_{max}(S^c)$. And,*
- *Let $\{p_i\}_{i \in A}$ and $\{t_i\}_{i \in A}$ be two distributions such that $p_i \geqslant t_i$ if and only if $i \in S$.*

Then $\sum_{i \in A} p_i X_i \geq \sum_{i \in A} t_i X_i$.

To see the lemma, notice that:

$$\begin{aligned}
\sum_{i \in A} p_i X_i &= \sum_{i \in S} p_i X_i + \sum_{i \in S^c} p_i X_i = \sum_{i \in S} t_i X_i + \sum_{i \in S} (p_i - t_i) X_i + \sum_{i \in S^c} t_i X_i + \sum_{i \in S^c} (p_i - t_i) X_i \\
&= \sum_{i \in A} t_i X_i + \sum_{i \in S} (p_i - t_i) X_i + \sum_{i \in S^c} (p_i - t_i) X_i .
\end{aligned}$$

However, $p_i - t_i$ is positive for $i \in S$ and negative otherwise, and $X_i \geq X_{max}(S^c)$ for $i \in S$ and $X_i \leq X_{max}(S^c)$ for $i \in S^c$. Thus,

$$\sum_{i \in A} p_i X_i \quad \geq \quad \sum_{i \in A} t_i X_i + \sum_{i \in S} (p_i - t_i) X_{max}(S^c) + \sum_{i \in S^c} (p_i - t_i) X_{max}(S^c) = \sum_{i \in A} t_i X_i \,.$$

◀

References

1 Masayuki Abe. Mix-networks on permutation networks. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT'99, pages 258–273, London, UK, UK, 1999. Springer-Verlag.

2 Masayuki Abe and Fumitaka Hoshino. Remarks on mix-network based on permutation networks. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 317–324. Springer Berlin Heidelberg, 2001.

3 David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

4 Artur Czumaj, Przemka Kanarek, Mirosław Kutyłowski, and Krzyztof Loryś. Delayed path coupling and generating random permutations via distributed stochastic processes. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280. Society for Industrial and Applied Mathematics, 1999.

5 Artur Czumaj, Przemka Kanarek, Krzysztof Lorys, and Miroslaw Kutylowski. Switching networks for generating random permutations, 2001.

6 Persi Diaconis and Mehrdad Shahshahani. On the eigenvalues of random matrices. *Journal of Applied Probability*, pages 49–62, 1994.

7 W T. Gowers. An almost m-wise independent random permutation of the cube. *Combinatorics Probability and Computing*, 5:119–130, 1996.

8 Shlomo Hoory, Avner Magen, Steven Myers, and Charles Rackoff. Simple permutations mix well. In *Automata, Languages and Programming*, pages 770–781. Springer, 2004.

9 Frank Thomson Leighton. *Introduction to parallel algorithms and architectures.* Morgan Kaufmann San Francisco, 1992.

10 Helger Lipmaa. Efficient nizk arguments via parallel verification of benes networks. In *To appear in SCN (9th Conference on Security and Cryptography for Networks) 2014*, 2014.

11 Nathan Lulov. *Random walks on the symmetric group generated by conjugacy classes.* PhD thesis, Harvard University, 1996.

12 Nathan Lulov and Igor Pak. Rapidly mixing random walks and bounds on characters of the symmetric group. *Journal of Algebraic Combinatorics*, 16(2):151–163, 2002.

13 Ben Morris. Improved mixing time bounds for the thorp shuffle and l-reversal chain. *The Annals of Probability*, pages 453–477, 2009.

14 Ben Morris, Phillip Rogaway, and Till Stegers. How to encipher messages on a small domain. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'09, pages 286–302, Berlin, Heidelberg, 2009. Springer-Verlag.

15 Charles Rackoff and Daniel R Simon. Cryptographic defense against traffic analysis. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 672–681. ACM, 1993.

Notes on Counting with Finite Machines

Dmitry Chistikov

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany
`dch@mpi-sws.org`

Abstract

We determine the descriptional complexity (smallest number of states, up to constant factors) of recognizing languages $\{1^n\}$ and $\{1^{tn} : t = 0, 1, 2, \ldots\}$ with state-based finite machines of various kinds. This task is understood as *counting* to n and modulo n, respectively, and was previously studied for classes of finite-state automata by Kupferman, Ta-Shma, and Vardi (2001). We show that for Turing machines it requires $\log n / \log \log n$ states in the worst case, and individual values are related to Kolmogorov complexity of the binary encoding of n. For deterministic pushdown and counter automata, the complexity is $\log n$ and $\sqrt{n}$, respectively; for alternating counter automata, we show an upper bound of $\log n$. For visibly pushdown automata, i. e., if the stack movements are determined by input symbols, we consider languages $\{a^n b^n\}$ and $\{a^{tn} b^{tn} : t = 0, 1, 2, \ldots\}$ and determine their complexity, of $\sqrt{n}$ and $\min(n_1 + n_2)$, respectively, with minimum over all factorizations $n = n_1 n_2$.

1998 ACM Subject Classification F1.1 Models of Computation, F4.3 Formal Languages

Keywords and phrases State complexity, Unary languages, Counting

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.339

1 Introduction

The task of counting is one of the most basic tasks that can be entrusted to computing devices. In the framework of formal language theory, it is natural to associate counting with finite machines that recognize singleton languages $\{1^n\}$, where $n = 0, 1, 2, \ldots$ Since even most primitive devices, namely finite automata, are expressive enough to perform this task, questions from the realm of descriptional complexity arise.

A standard problem setting here can be traced back to a classic 1971 paper by Meyer and Fischer [16] and can be stated as follows: given a specific class $\mathcal{C}$ of computational machines (such as nondeterministic finite automata), estimate the function $f(n)$, whose value on an arbitrary non-negative integer n is defined as the smallest descriptional complexity of a machine from this class that recognizes the language $\{1^n\}$. One then says that machines from $\mathcal{C}$ *count to* n with complexity $f(n)$. All classes are usually expressive enough to contain at least one appropriate machine for each n, and so it is always the case that $f(n) < \infty$.

To the best of our knowledge, the most thorough account of the problems of counting with finite machines can be found in a technical report from 2001 by Kupferman, Ta-Shma, and Vardi [12], which coins the term *counting to* n to refer to the problem of recognizing the language $\{1^n\}$. Kupferman et al. study this problem for standard classes of finite machines: deterministic, nondeterministic, universal, and alternating automata, and characterize the smallest number of states sufficient to count to n within each of these classes. Disregarding constant factors, this number is n for deterministic and nondeterministic automata, $\sqrt{n}$ for universal and $\log n$ for alternating automata.

Our contribution

Our primary focus is on the problems of counting with more powerful devices, capable of recognizing non-regular languages. As far as we know, these problems have never been addressed systematically. Observe that although one of the goals of the study of descriptional complexity is to compare the expressive power of machines from different classes, the notion of complexity in the definition above is tied to a specific class of machines. As a consequence, for different classes one has to use "matching" complexity measures. For all classes of machines considered in this paper, we use measures that generalize state complexity of finite automata.

Our results are as follows. First, for Turing machines with tape alphabet $\{0,1\}$, we show in Section 3 that the complexity of counting is at most $\log n/\log\log n$. As a corollary, for an arbitrary string s (not necessarily over the one-letter alphabet), the state complexity of generating it with a Turing machine is asymptotically equal to $K(s)/(|\Delta|-1)\log K(s)$, where $K(\cdot)$ denotes the standard Kolmogorov complexity and Δ is the tape alphabet. (For $|\Delta| \geq 3$, this follows from a construction by Chaitin [4, sections 1.5–1.6], and we fill in the gap for $|\Delta| = 2$.) In other words, whenever there exists a Turing machine that produces s when run on empty tape and has binary encoding of length k, there always exists another Turing machine with the same properties that has at most $k/(|\Delta|-1)\log k$ states, and the latter bound is asymptotically tight.

Second, we study the problems of counting for classes of pushdown automata (PDA). For deterministic PDA, we show in Section 4 that matching upper and lower bounds of $O(\log n)$ and $\Omega(\log n)$ for all n follow from earlier results due to Pighizzini [19] and Chistikov and Majumdar [7]. Subclasses of PDA, however, require separate consideration. For deterministic counter automata, i. e., when the stack alphabet contains only one symbol apart from the bottom-of-stack, we obtain an upper bound of $O(\sqrt{n})$ and show a matching lower bound $\Omega(\sqrt{n})$ by reducing to counting with deterministic finite automata (DFA). For alternating counter automata, we prove an upper bound of $O(\log n)$ (here we use a complexity measure that is more refined than just the number of states, so this result is not subsumed by the fact, due to Kupferman et al., that alternating finite automata count to n with $\lceil \log n \rceil$ states).

Third, we consider another well-known subclass of PDA called visibly pushdown automata (VPA, also known as nested word automata and input-driven PDA) in Section 5. While counting (and, indeed, recognizing any unary language) for VPA is easily shown to be as hard as for finite automata, we prove that deterministic VPA recognize languages $\{a^n b^n\}$—another problem interpreted as counting—with complexity $\Theta(\sqrt{n})$. We also show that the complexity of recognizing languages $\{a^{tn} b^{tn} \colon t = 0, 1, 2, \ldots\}$ is $\Theta(\min(n_1 + n_2))$, where the minimum is taken over all factorizations $n = n_1 n_2$; this function ranges, for different values of n, between $\Theta(\sqrt{n})$ and $\Theta(n)$. By showing sequences for which its value is $o(n)$, we refute a conjecture of Salomaa [22] on the state complexity, with respect to VPA, of a related language.

Acknowledgements. This paper would be very different without many people's help. I would like to thank Rupak Majumdar for support and for sharing a reference to an earlier version, from 1999, of the technical report by Kupferman, Ta-Shma, and Vardi [12]. I am grateful to Alexander Shen, who pointed me to Chaitin's work [4] upon hearing the result of Corollary 2. Abuzer Yakaryılmaz asked a question on StackExchange that let me come up with a construction of an alternating counter automaton recognizing a non-regular language $\{1^n \colon n = 2^k,\ k = 0, 1, 2, \ldots\}$ [3] and lead to Theorem 5. I would also like to thank Rose Hoberman for useful discussions.

2 Related work

As explained above, the starting point for our work is the technical report of Kupferman, Ta-Shma, and Vardi [12], who build upon the results of Leiss [13], Birget [2], and Chrobak [8]. From the general language-theoretic perspective, the problems of counting are closely tied to numerous phenomena of unary languages, that is, languages over a one-letter alphabet (also known as tally languages). Classes of unary languages possess many properties that cannot be observed in classes of languages over larger alphabets. Perhaps the most widely known is the theorem saying that every unary context-free language is regular, first proved by Ginsburg and Rice in 1962 [11].

Although the languages $\{1^n\}$ studied in the present paper are finite and therefore regular, we are interested in their descriptional complexity with respect to classes of machines that recognize, in general, non-regular languages. Hence, related are not only descriptional complexity questions for machines specifying non-regular sets, but also questions of so-called succinctness of representations, or economy of description. This term is associated with the following question, first asked by Meyer and Fischer in 1971 [16]: suppose that some language L belongs to a certain class $\mathcal{C}$; how short a description can this language L have with respect to $\mathcal{C}$, compared to the shortest description within some specific subclass $\mathcal{C}' \subsetneq \mathcal{C}$ such that $L \in \mathcal{C}'$? Or, in other words, what is the largest blowup that can be observed when translating a description from the class $\mathcal{C}$ into the "terms" of $\mathcal{C}'$?

For general context-free and regular languages (with sizes of context-free grammars—CFG—and deterministic finite automata—DFA—as complexity measures), the answer to this question is given by Meyer and Fischer: the description of a DFA cannot be bounded by any recursive function in the size of a CFG. For unary languages, translations from CFG (in Chomsky normal form) into NFA and DFA are shown to be at most exponential [20].

One can easily see that using PDA instead of CFG also leaves the translation exponential. Tight bounds on the size of the blowup in the case of unary nondeterministic PDA are also given in [20], and the unary deterministic case is studied in [19]. A general connection between unary deterministic PDA and grammar compression of binary words is established in [7]. In the present paper, we use these results to obtain bounds on the size of PDA.

In general, descriptional complexity problems for general context-free languages per se have so far been mostly associated with grammars and not with automata. As pointed out above, however, PDA have been intensively studied in connection with questions of economy of description. In this area, we wish to highlight the paper [10], which not only continues the line of research started by Meyer and Fischer, but also proves exponential lower bounds for descriptional complexity of PDA recognizing several natural finite languages.

For context-free grammars, we refer the reader to a recent paper [9], which obtains strong lower bounds on the size of CFG for several specific finite languages. Counting problems for context-free grammars are tightly related to the well-studied concept of an addition chain (see, e. g., [6, Section V-B]), which is a restriction, to the unary alphabet, of a context-free grammar that generates a single word. (A more general topic is, of course, grammar compression of non-unary words, see also [21, 6, 15].) As an illustrative example, one can easily show that the language $\{1^n\}$ can be generated by a context-free grammar with at most $O(\log n)$ symbols in right-hand sides of productions, and prove that this bound is tight.

For the subclass of context-free languages recognized by visibly pushdown automata, descriptional complexity questions have been studied somewhat more extensively; see, e. g., [17, 18], where these machines are called "input-driven pushdown automata".

As for descriptional complexity for Turing machines, the number of states as a complexity measure was studied by Chaitin [4]; his paper initiated the study of what is now known as Kolmogorov complexity [14], but then the focus very quickly shifted towards a different measure, the length of a binary string that encodes the description of a machine. In our work, we fill in the remaining gap for the original measure, i. e., for the number of states.

3 Turing machines

A *Turing machine* (TM) with input alphabet Σ and tape alphabet $\Delta \supseteq \Sigma$ is a tuple $\mathcal{M} = (\Sigma, \Delta, \Box, Q, q_0, H, \delta)$, where $\Box \in \Delta \setminus \Sigma$ is the blank symbol, Q the set of states, $q_0 \in Q$ the initial state, H the set of halting states, and $\delta \subseteq (Q \times \Delta) \times (Q \times \Delta \times \{-1, 0, +1\})$ the transition relation.

Configurations of $\mathcal{M}$ have the form (q, μ, n), where $q \in Q$, $\mu\colon \mathbb{Z} \to \Delta$, and $n \in \mathbb{Z}$; the interpretation is that cells of the infinite tape, indexed by $\mathbb{Z}$, contain symbols from Δ specified by μ, with $|\mu^{-1}(\Delta \setminus \{\Box\})| < \infty$, the control state of the machine is q, and the head of the machine is at the nth cell of the tape. If $\mathcal{M}$ is run on input $w \in \Sigma^*$, then the initial configuration is $(q_0, \mu_w, 0)$ where $\mu_w(i)$ is $w[i]$ for $0 \le i < |w|$ and $\Box$ otherwise; halting configurations are those with $q \in H$. The transition relation imparts a step-reachability on configurations: if $(q, \sigma, q', \tau, d) \in \delta$ and $q \notin H$, then at a configuration (q, μ, n), if $\mu(n) = \sigma$, the machine can overwrite this σ with τ, change its control state to q', and change the head's position by d. A machine is *deterministic* if from each configuration at most one configuration is reachable in one step (i. e., δ defines a mapping from $Q \times \Delta$ to $Q \times \Delta \times \{-1, 0, +1\}$).

Instead of the problem of *recognizing* the language $\{1^n\}$ with a Turing machine, we consider the problem of *generating* this language; our results can be easily extended to language recognition as well. A machine $\mathcal{M}$ *outputs* a word $u \in (\Delta \setminus \{\Box\})^*$ when run on $w \in \Sigma^*$ if from $(q_0, \mu_w, 0)$ it reaches some configuration (q, μ, n) with $q \in H$ such that $\mu^{-1}(\Delta \setminus \{\Box\}) = [a, a + |u| - 1]$ for some $a \in \mathbb{Z}$ and $\mu(a + i) = u[i]$ for $0 \le i < |u|$.

Let $Q_\Delta(s)$, $s \in \Sigma^* \subsetneq \Delta^*$, be the smallest number m such that there exists a deterministic TM with m states and tape alphabet Δ that outputs s when run on empty tape. The interpretation is that TMs with tape alphabet Δ count to n with complexity $Q_\Delta(1^n)$.

▶ **Theorem 1.** *Fix* $\Sigma = \{1\}$, $\Delta = \{0, 1\}$, *and* $\Box = 0$. *Then* $Q_\Delta(1^n) \lesssim \log n / \log\log n$.[1]

Proof. We construct, for each n, a Turing machine $\mathcal{M}_n$ with tape alphabet $\{0, 1\}$ that has $\log n / \log\log n\,(1 - o(1))$ states and outputs 1^n when run on empty tape. Let us show the workings of $\mathcal{M}_n$. Let $n = \sum_{i=0}^{l-1} b_i k^{l-1-i}$ be the k-ary representation of n, with $b_i \in \{0, 1, \ldots, k-1\}$. The value of k will be fixed later. The TM will keep two counters on the tape, denoted i and v, and written out in unary, as 1^i and 1^v respectively. Initially $v = 0$ and $i = 0$. Next, the machine will get into the main loop and do the following, as long as $i < l$: multiply v by k, add b_i to v, and increment i. For multiplication, two auxiliary counters u and j are needed: at first $j = k$ and $u = 0$, then the machine will add v to u and decrement j as long as $j > 0$; when j becomes 0, the value of u takes the place of v. In the main loop, when i becomes l, the TM will terminate with output v.

It is easy to see that this procedure is correct: $v = n$ at the end of the computation. Let us count the number of states needed to implement it. For initial setup, $O(1)$ states suffice. In the main loop, all operations are fixed and take up $O(1)$ states, with two exceptions: one of setting j to k, and another of choosing (and generating) b_i according to i (we also include

[1] We write $f(n) \lesssim g(n)$ iff $f(n) \le g(n) \cdot (1 + o(1))$.

termination on $i = l$ into this exception). For the former, $k + O(1)$ states suffice; we focus on the latter next. (We omit the description of how to keep all counters on the tape, because this can be done with standard techniques.)

The choice of b_i is implemented as follows: suppose on the tape there are only blanks (i. e., 0s) to the right of the counter i. This counter is kept as 1^i. Create a set of $l + k$ states of $\mathcal{M}_n$, denoted $q_0, q_1, \ldots, q_{l-1}$ and $p_0, \ldots, p_{k-1}$ (here q_0 is *not* the initial state of $\mathcal{M}_n$). Let the machine transition to q_0 and step on the leftmost 1 in 1^i. At each of the states q_j, $j < l - 1$, if the observed cell tape contains a 1, the machine moves its head one cell to the right and goes to q_{j+1}. If the observed cell is 0, then the value of the counter is j. In this case, $\mathcal{M}_n$ also moves its head one cell to the right and goes to p_s for $s = b_j \in \{0, 1, \ldots, k-1\}$. At q_{l-1}, if the machine observes 1, it invokes the termination procedure as $i = l$; otherwise, it moves its head one cell to the right and goes to p_s for $s = b_{l-1}$. All these transitions are hardwired in $\mathcal{M}_n$ and encode, when taken together, the number n. At each of the states p_s, $0 < s \le k - 1$, the machine writes a 1, moves its head one cell to the right, and goes to p_{s-1}; there is no need to write anything at the state p_0. As a result, if put in q_0, the machine will arrive at the state p_0 with the unchanged counter i and with 1^{b_i} written to the right of 1^i and separated from it by a 0. This is indeed the desired result; the number of states used in this gadget is $l + k + O(1)$.

In total, $\mathcal{M}_n$ has $|Q| \le l + 2k + O(1)$ states. Notice that $l = \lceil \log_k(n+1) \rceil = \log n / \log k + O(1)$ and take $k = \log n/(\log\log n)^2$, then $2k = o(\log n/\log\log n)$ as $n \to \infty$, and so $|Q| \le \log n/\log\log n\,(1 - o(1))$, which completes the proof. ◀

Corollary 2 below shows that the upper bound given by Theorem 1 is tight, i. e., that $\limsup Q_{\{0,1\}}(n)/(\log n/\log\log n) = 1$. However, the lower bound of $\log n_k/\log\log n_k\,(1 + o(1))$ only holds for certain sequences of naturals, and not for all $n \ge 0$.

For individual values of n, we relate the complexity of counting to Kolmogorov complexity; the following definitions are standard [14]. If some Turing machine $\mathcal{M}$, when given input y, halts and outputs x, then the string y is called a *description* of x with respect to $\mathcal{M}$. Let us also fix some *universal* Turing machine $\mathcal{U}$; given input (z, y), the TM $\mathcal{U}$ runs the machine described by z on input y. Here we fix, in advance, some descriptional system for Turing machines in which strings $z \in \{0,1\}^*$ encode machines. Define the *Kolmogorov complexity* of a string $x \in \{0,1\}^*$ as the smallest number k such that x has some description of size k, $y \in \{0,1\}^k$, with respect to $\mathcal{U}$.

It follows from results of Chaitin [4, sections 1.5–1.6] that for any alphabets $\Sigma \subsetneq \Delta$, $|\Delta| \ge 3$, and all strings $s \in \Sigma^*$ it holds that $Q_\Delta(s) \sim K(s)/(|\Delta| - 1)\log K(s)$ as $K(s) \to \infty$; Theorem 1 allows us to extend this result to a quite different case $|\Delta| = 2$ (where, unlike Chaitin, we have to use unary-encoded numbers). By $\mathrm{bin}(n)$ we denote the binary encoding of a non-negative integer n; that is, a word over the alphabet $\{0,1\}$.

▶ **Corollary 2.** *Under conditions of Theorem 1, for all $n \ge 0$ it holds that $Q_{\{0,1\}}(1^n) \sim K(1^n)/\log K(1^n) \sim K(\mathrm{bin}(n))/\log K(\mathrm{bin}(n))$ as $K(\mathrm{bin}(n)) \to \infty$.*

Another way of stating the last result is as follows: whenever there exists a Turing machine, with tape alphabet Δ, $|\Delta| \ge 2$, that produces a string s when run on empty tape and has an encoding of length k, there always exists another Turing machine with the same properties that has at most $k/(|\Delta| - 1)\log k \cdot (1 + o(1))$ states; in other words, encodings of Turing machines can always be fit into the smallest possible number of states.

▶ Remark. The lower bound of $K(\mathrm{bin}(n))/\log K(\mathrm{bin}(n))$ holds for all classes of machines considered in this paper (including nondeterministic and alternating ones), provided that the number of transitions leaving each control state is bounded by a constant.

4 Pushdown and counter automata

A *pushdown automaton* (PDA) over an input alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, q_0, P, \bot, F, \delta)$, where Q is the set of (control) states, $q_0 \in Q$ the initial state, P the set of stack symbols, $\bot \in P$ the bottom-of-the-stack symbol, $F \subseteq Q$ the set of accepting states, and $\delta \subseteq (Q \times P \times (\Sigma \cup \{\varepsilon\})) \times (Q \times P^{\leq 2})$ the transition relation.

Configurations of a PDA $\mathcal{A}$ are tuples of the form (q, s, u), where $q \in Q$ is the current state, $s \in (P \setminus \{\bot\})^* \bot$ the contents of the stack, and $u \in \Sigma^*$ the remaining input tape. For an input word $w \in \Sigma^*$, the initial configuration is $(q_0, \bot, w)$. For every transition $(q, p, \sigma, q', t) \in \delta$, the PDA can, consuming σ from the input, move from a configuration (q, s, u) to a configuration (q', s', u') such that $u = \sigma u'$, $s = pv$ and $s' = tv$ for some $v \in P^*$. If $p = \bot$, then $t \in (P \setminus \{\bot\})^{\leq 1} \bot$, otherwise $t \in (P \setminus \{\bot\})^{\leq 2}$.

The PDA *accepts* a word $w \in \Sigma^*$ if from $(q_0, \bot, w)$ it can reach a configuration (q, s, ε) with $q \in F$. The PDA is *deterministic* (DPDA) if from each configuration at most one configuration is reachable in one step; it suffices that, first, for each $q \in Q$ transitions $(q, p, \sigma, q', t) \in \delta$ either all have $\sigma = \varepsilon$ or $\sigma \neq \varepsilon$, and, second, that δ defines a partial mapping from $Q \times P \times (\Sigma \cup \{\varepsilon\})$ to $Q \times P^{\leq 2}$.

The product $|Q| \cdot |P|$ shall be called the *complexity* of a deterministic PDA. We recall that machines from a certain class are said to *count to n* with complexity $f(n)$ if $f(n)$ is the smallest size of a machine from the class that recognizes the language $\{1^n\}$; for a related language $\{1^{tn} : t = 0, 1, 2, \ldots\}$, we use the term *counting modulo n*.

▶ **Theorem 3.** *Deterministic pushdown automata count to n and modulo n with complexity $\Theta(\log n)$.*

This result can be obtained as an application of Theorem 1 in Chistikov and Majumdar [7]. The theorem states that the size of the smallest DPDA recognizing a language $R \subseteq \{1\}^*$ is within a constant factor of the smallest size of a pair of context-free grammars $\mathcal{P}$, $\mathcal{L}$ generating single words $\mathrm{eval}(\mathcal{P})$, $\mathrm{eval}(\mathcal{L})$ such that the ith element of the sequence $c = \mathrm{eval}(\mathcal{P}) \cdot (\mathrm{eval}(\mathcal{L}))^\omega$ is 1 if $1^i \in R$ and 0 if $1^i \notin R$. In our case $c = 0^n \cdot 1 \cdot 0^\omega$ for $R = \{1^n\}$ and $c = (10^{n-1})^\omega$ for $R = \{1^{tn} : t = 0, 1, 2, \ldots\}$; in both cases, logarithmic upper and lower bounds on the size of $(\mathcal{P}, \mathcal{L})$ follow, because $\Theta(\log n)$ constant-size productions are necessary and sufficient for a CFG to generate a word of length n. The lower bound can also be deduced, under minor structural assumptions about DPDA, from an earlier result of Pighizzini [19, Theorem 12].

A *counter automaton* (CA) is a pushdown automaton with $|P| = 2$, that is, with just one stack symbol apart from $\bot$. It is convenient to think of a CA as of a finite automaton with a non-negative integer counter. Available operations on the counter are increment, decrement, and zero test; the CA cannot directly distinguish between different non-zero counter values.

▶ **Theorem 4.** *Deterministic counter automata count to n and modulo n with complexity $\Theta(\sqrt{n})$.*

Proof. We first prove the upper bound. Denote $r = \lfloor \sqrt{n} \rfloor$. Let a deterministic counter automaton $\mathcal{A}_n$ first consume $n - r^2 \leq 2r$ letters from the input and then increase the counter value to r; these operations require at most $3r + 1$ states, the last of which we denote by q. At this state q, the automaton $\mathcal{A}_n$ performs a zero test on the counter: if its value is zero, $\mathcal{A}_n$ accepts and terminates; otherwise it decrements the counter, consumes r letters from the input, and returns to q. It is easy to see that $\mathcal{A}_n$ has $4r + O(1)$ states and recognizes $\{1^n\}$; if instead of termination it goes to the initial state, it will recognize $\{1^{tn} : t = 0, 1, 2, \ldots\}$.

To prove the lower bound, consider any deterministic counter automaton $\mathcal{A}$. Suppose its set of states is Q, with $|Q| = m$. Construct an auxiliary device: a deterministic finite automaton $\mathcal{D}$ with states of the form (q, k), for $q \in Q$ and $0 \leq k \leq m + 1$. Direct the transitions of $\mathcal{D}$ in such a way that $\mathcal{D}$ simulates $\mathcal{A}$, keeping the value of the counter in the component of the control state denoted k; do not add transitions from states $(q, m + 1)$ that need to increase the counter.

If $\mathcal{D}$ recognizes the same language as $\mathcal{A}$, then the desired bound holds, because DFA—even with ε-transitions—need n states to count both to and modulo n. Now consider the only alternative: suppose that a computation of the CA $\mathcal{A}$ on some input word 1^s gets to a state q with counter value $m + 1$ and then increases the counter. For each $i = 1, \ldots, m + 1$ consider, in this computation, the last configuration before this point when the counter value is i. Denote these configurations by (q_i, i); they occur in the computation in the order with $i = 1$ first and $i = m + 1$ last. Note that between $(q_1, 1)$ and $(q_{m+1}, m + 1)$ the counter value is always positive, and recall that the automaton cannot distinguish between different positive values. By pigeonhole principle, there exist indices $j < k$ such that $q_j = q_k$, so $\mathcal{A}$ essentially gets into a loop: if the input tape provides it with infinitely many 1s, then it will be following the transitions of this loop forever. In $\mathcal{D}$, rerouting to $(q_k, j) = (q_j, j)$ the transitions with destination (q_k, k) will make it simulate $\mathcal{A}$ faithfully, i. e., recognize the same language.

Let us now consider each counting task separately. First suppose that $\mathcal{A}$ recognizes the language $\{1^n\}$, then either the loop we have found does not contain any accepting state, or it contains an accepting state but does not consume any letters from the input. (Indeed, if neither of these two conditions held, then $\mathcal{A}$ would accept infinitely many words.) In both cases it is possible to replace this loop with at most one state in the DFA. But then the obtained DFA, possibly with ε-transitions, will accept $\{1^n\}$, so it should contain at least $n + 1$ states. As a result, $m(m + 2) \geq n + 1$, and the desired bound follows.

Now suppose that $\mathcal{A}$ recognizes $\{1^{tn} : t = 0, 1, 2, \ldots\}$. Since the constructed DFA, like $\mathcal{A}$, is trapped forever in the loop, the states in the loop should enable it to accept input words 1^{tn} and reject all other input words. This means that the loop should consume at least n letters from the input; since it contains at most $m^2 + 1$ transitions, we conclude that $n \leq m^2 + 1$. This completes the proof. ◀

We next consider the *alternating* version of counter automata; we use the standard definition of alternation, which is a little different from the one that appeared originally in Chandra, Kozen, and Stockmeyer [5]. Informally, the machine is extended with the ability to make guesses and, dually, assertions during the computation.

More formally, suppose each state $q \in Q$ is labeled with either $\exists$ or $\forall$; consider the computational tree imparted on configurations by step-reachability. A node in this tree is *accepting* if, first, its state q belongs to F, or, second, its state is labeled by $\exists$ and has an accepting successor, or, third, its state is labeled by $\forall$ and all its successors are accepting (we use the least fixpoint here; i. e., only finite branches can count towards acceptance). An input word is *accepted* if the root in the computational tree is accepting.

Note that whenever the syntax of a state-based machine allows an unbounded number of transitions from a particular state (as sometimes needed by, e. g., nondeterministic and alternating machines), the number of control states is no longer a good complexity measure. For instance, alternating finite-state automata counting to n described by Kupferman et al. [12] have $O(\log n)$ states but $\Omega(\log^2 n)$ transitions. To avoid this issue, we use the number of transitions in δ as the complexity measure for alternating counter automata.

▶ **Theorem 5.** *Alternating counter automata count to n with complexity $O(\log n)$.*

Proof. Suppose $n = \sum_{i=0}^{l-1} b_i 2^i$, with $l = \lceil \log_2(n + 1) \rceil$ and all $b_i \in \{0, 1\}$, and create control states $q_0, q_1, \ldots, q_l$. The state q_0 will be the initial state of the automaton $\mathcal{A}$. Starting at a state q_i, $i < l$, with a zero counter value, $\mathcal{A}$ will, if $b_i = 1$, consume a letter from the input tape, and then, regardless of the value of b_i, get into a loop where it simultaneously reads from the input and increases the value of the counter. It can nondeterministically choose to exit the loop; the interpretation is that it guesses some value m and, with m iterations, reads 1^m from the input and increases the value of the counter to m. Upon exiting the loop, $\mathcal{A}$ branches universally: the first branch verifies that the value of the counter is equal to the number of remaining letters on the input tape, and the second branch goes to q_{i+1}. At the state q_l, $\mathcal{A}$ accepts. It is easy to see that $\mathcal{A}$ indeed recognizes the language $\{1^n\}$; $O(1)$-bounded branching ensures that $\mathcal{A}$ has $O(\log n)$ transitions. ◀

5 Visibly pushdown automata

A *visibly pushdown automaton* [1], or a VPA, $\mathcal{A}$ is a pushdown automaton that has the following property: there exists a partition of the input alphabet Σ into three parts, $\Sigma = \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}} \cup \Sigma_{\mathsf{int}}$, such that $\mathcal{A}$ always pushes a symbol upon reading a letter from Σ_{call}, pops a symbol upon reading a letter from Σ_{ret}, and does not use the stack on letters from Σ_{int}. It is implied that, in a VPA, destinations of transitions driven by letters from $\Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{int}}$ cannot depend on the top symbol of the stack. Furthermore, a VPA is not allowed to have ε-transitions—i. e., all transitions are required to consume at least one letter from the input.

It is easy to see that in the problem of counting, defined as recognizing languages $\{1^n\}$ and $\{1^{tn} \colon t = 0, 1, 2, \ldots\}$, visibly pushdown automata cannot do better than standard finite automata. So it is natural to consider different, although closely related languages $\{a^n b^n\}$ and $\{a^{tn} b^{tn} \colon t = 0, 1, 2, \ldots\}$, where $n = 0, 1, 2, \ldots$, with the restriction that $a \in \Sigma_{\mathsf{call}}$ and $b \in \Sigma_{\mathsf{ret}}$. Basically, we convert "linear" input words over Σ into nested words, as in [1], which enables our automata to use the stack. Note that in the framework of nested words, our languages $\{a^n b^n\}$ and $\{a^{tn} b^{tn} \colon t = 0, 1, 2, \ldots\}$ translate to languages over a one-letter alphabet, that is, to a unary language of nested words.

Salomaa [22] proved that every deterministic visibly pushdown automaton over the alphabet $\Sigma = \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$, with $\Sigma_{\mathsf{call}} = \{a_1, a_2\}$ and $\Sigma_{\mathsf{ret}} = \{b_1, b_2\}$, recognizing a related language

$$L_n = \{a_{i_1} \ldots a_{i_k} b_{i_k} \ldots b_{i_1} \colon i_1, \ldots, i_k \in \{1, 2\},\ k = tn,\ t = 0, 1, 2, \ldots\}, \tag{1}$$

should have at least $\sqrt{n}$ states. The proof does not essentially use different kinds of push and pop symbols, so the lower bound extends to $\{a^{tn} b^{tn} \colon t = 0, 1, 2, \ldots\}$. Moreover, the crucial point is just that the word $a^n b^n$ is accepted, and all words $a^m b^m$, $0 < m < n$, rejected; as a result, the lower bound of $\sqrt{n}$ holds for the language $\{a^n b^n\}$ as well.

Salomaa also constructed a VPA with $n + 1$ states and 3 stack symbols for L_n, and conjectured that this VPA is optimal in terms of $|Q| + |P|$. We refute this conjecture by constructing a VPA with $O(\sqrt{n})$ states and $O(1)$ stack symbols that recognizes L_n.

▶ **Theorem 6.** *For every $n \geq 1$, there exists a deterministic visibly pushdown automaton $\mathcal{A}$ that recognizes the language $\{a^n b^n\}$ and satisfies $|Q| \leq O(\sqrt{n})$ and $|P| \leq O(1)$. In contrast, every (possibly nondeterministic) visibly pushdown automaton that recognizes this language satisfies the inequality $|Q| \geq \sqrt{n}$.*

▶ **Theorem 7.** *If $n = n_1 n_2 \geq 1$, then there exists a deterministic visibly pushdown automaton $\mathcal{A}$ that recognizes the language $\{a^{tn} b^{tn} \colon t = 0, 1, 2, \ldots\}$ and satisfies $|Q| \leq O(n_1 + n_2)$ and $|P| \leq O(1)$. In contrast, every deterministic visibly pushdown automaton that recognizes this language satisfies the inequality $|Q| \geq \Omega(\min(n_1 + n_2))$, where the minimum is over all factorizations $n = n_1 n_2$ with natural $n_1, n_2 \geq 1$.*

Theorem 7 shows that the smallest VPA counting modulo n have $\Theta(f(n))$ control states, with $f(n) = \min(n_1 + n_2)$ over all factorizations $n = n_1 n_2$; here $\sqrt{n} \le f(n) \le n + 1$ for all n, the left-hand inequality is tight for perfect squares, $n = k^2$, and the right-hand inequality for prime numbers, $n = p$.

▶ **Proposition 8.** *The results of Theorem 7 also hold for the language L_n, as defined by* (1). *For $n = k^2$, $k = 1, 2, \ldots$, the automaton witnessing the upper bound satisfies $|Q| + |P| = O(k) = O(\sqrt{n})$, which disproves the conjecture of Salomaa* [22]. *For $n = p$, p prime, the lower bound shows that the construction of VPA for L_n given in* [22] *is optimal up to a constant factor.*

The rest of the section is devoted to the proofs of Theorems 6 and 7 and Proposition 8.

5.1 Main ingredients for upper bounds

The high-level idea for proving the upper bounds is simple and common for all three languages: let the automaton first count blocks of a, each of size n_1, and mark their starting positions by pushing a special symbol on the stack. Then, as the automaton reads bs and pops from the stack, it counts the number of blocks. Here it can count either to n_2 or modulo n_2, and the bound on the total number of states is $O(n_1 + n_2)$. So we will need an auxiliary construction, that of a VPA recognizing the language

$$L_{\mathsf{cycles}} = \{a^s b^{t+d} \colon n_1 \mid s,\ n_1 n_2 \mid t,\ s \ge t + d,\ 0 \le d < n_1\}. \tag{2}$$

Let us define such a VPA, denoted $\mathcal{V}(n_1, n_2)$, as follows.

First take n_1 control states and connect them in a single cycle such that reading an a from the input moves the device one step along the cycle. Make one of these states, q_1, the initial state and let the device push 1 when it reads a at q_1, and push 0 when it reads a at any other state of the cycle. Now take new n_2 control states and connect them into a different cycle. The automaton will only use these states when reading bs from the input; more specifically, demand that popping 0 from the stack not change the control state, and popping 1 moves the device one state further along the cycle.

Now connect these two disjoint cycles in the following manner. Let the device, upon reading b and popping 0 at q_1, move to some specific state of the second cycle, q_2. We shall call this state q_2 the entry point of the second cycle. Make q_1 and q_2 the only final states, and turn all missing transitions into some (new) sink state.

▶ **Lemma 9.** *The VPA $\mathcal{V}(n_1, n_2)$ has $n_1 + n_2 + O(1)$ control states and* 3 *stack symbols and recognizes L_{cycles}.*

5.2 Proof of Theorem 6

As mentioned above, the lower bound is essentially shown in Salomaa [22, Theorem 4.1]; and it is instructive to see that it holds even for nondeterministic VPA: it suffices to carry out the reasoning for an accepting branch of the computation. So, in what follows we only need to focus on the upper bound.

Given n, choose $n_1 = n_2 = \lfloor \sqrt{n-1} \rfloor$ and $r = n - n_1^2 > 0$, construct the VPA $\mathcal{V}(n_1, n_2)$ described above, and modify it in the following way. First, add a new stack symbol 2, make a simple path of r transitions reading a and pushing 2, and attach it to the VPA so that the last of these transitions leads to q_1, the "entry point" of the first cycle in $\mathcal{V}(n_1, n_2)$. Mark the source of the attached path, q_0, as the initial state of the VPA. Next, consider the second cycle of the VPA and transform it into a path by changing the destination of the transition that pops 1 and leads from a state q to q_2, the "entry point" of the second cycle. Make this transition lead from q to a new state q' instead, and then add to q' a path of r transitions that read b from the input and pop 2 from the stack. Make the last state on this path the only final state. (As previously, all missing transitions are sent to the sink state.) Now it is easy to see that the obtained VPA accepts exactly the words $a^s b^{s'}$ with $s = r + kn_1$, $s' = n_1^2 + r$, and $s = s'$—or, to put it differently, recognizes the language $\{a^n b^n\}$. Since it has $n_1 + n_2 + 2r + O(1) \le 4\sqrt{n} + O(1)$ control states and 4 stack symbols, this completes the proof of Theorem 6.

5.3 Main ingredients for lower bounds

We now turn our attention to the lower bound of Theorem 7, for the language $\{a^{tn} b^{tn} : t = 0, 1, 2, \ldots\}$. Our goal is to show that every deterministic VPA recognizing this language has at least $\Omega(\min(n_1 + n_2))$ states, where min is taken over all factorizations $n = n_1 n_2$ with integers $n_1, n_2 \ge 1$.

The key insight is the decomposition of a VPA for this language into two finite-state transducers. The first transducer transforms the first part of the input into a sequence of stack symbols, and the second transducer transforms the reversal of this sequence into a sequence of accepting and non-accepting states. Periodic behaviour of these transducers enables us to obtain the desired lower bound.

To use this idea, we need to go through some auxiliary constructions first. Take a (deterministic) VPA $\mathcal{A}$ and denote by Q the set of its control states. First consider the behaviour of $\mathcal{A}$ on words of the form a^s, $s = 0, 1, 2, \ldots,$—one can regard them all as prefixes of the infinite sequence a^ω. Restricted to these words, the VPA $\mathcal{A}$ behaves as a deterministic letter-to-letter finite-state transducer, with symbols pushed on the stack interpreted as output letters. Since the input to this *push-transducer* is a prefix of the infinite sequence a^ω, it is easy to see that the sequence of the states $\mathcal{A}$ gets into is eventually periodic with some period $k \le |Q|$, and the word pushed on the stack is of the form $uw^m w'$, where $|w| = k$ and $w = w'w''$ for some w''. That is, the push-transducer transforms a^ω into uw^ω; words u, w, and w' are uniquely determined by q, the state in which $\mathcal{A}$ arrives upon reading a^s. Denote by Q' the set of states that $\mathcal{A}$ visits infinitely often; $|Q'| = k$.

Now let $\mathcal{A}$ stop in some state $q \in Q'$ after an input a^s. Suppose this state q is fixed. If, from this point on, the input tape supplies $\mathcal{A}$ with symbols b only, then $\mathcal{A}$ will operate as another finite-state transducer, the *pop-transducer*: the symbols popped from the stack are interpreted as input letters of the transducer, and the states from Q the device visits are the output letters. Observe that the input to this transducer is $(\bar{w})^m \bar{u}$, for $\bar{w} = (w')^R (w'')^R$ and $\bar{u} = (w')^R u^R$, where by v^R we denote the reversal of a word v. This input is a prefix of the infinite periodic sequence $\alpha_q = (\bar{w})^\omega$ augmented with the finite word $\bar{u}$. Note that the words $\bar{w}$ and $\bar{u}$ are uniquely determined by q.

▶ **Lemma 10.** *For each $q \in Q'$, the pop-transducer transforms the sequence α_q into an eventually periodic sequence with a period r_q such that $r_q \mid t_q k$ for some $t_q \le |Q|$.*

▶ **Lemma 11.** *For each i, $0 \le i < k$, the behaviour of $\mathcal{A}$ on words $a^s b^s$ with $s \bmod k = i$ is eventually periodic with period r_q for some fixed $q \in Q'$: more precisely, for each i, $0 \le i < k$, there exists some $q \in Q'$ and an integer $s^{(i)}$ such that the VPA $\mathcal{A}$ either accepts or rejects both words $a^{s_1} b^{s_1}$ and $a^{s_2} b^{s_2}$, provided that $s_1, s_2 \ge s^{(i)}$, the period r_q divides $|s_1 - s_2|$, and k divides $|s_1 - s_2|$.*

▶ **Lemma 12.** *If $\mathcal{A}$ recognizes $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$, then $n \mid g_1 g_2$ for some integers $g_1, g_2 \le |Q|$.*

As the proof shows, the conclusion of Lemma 12 holds for any VPA $\mathcal{A}$ that recognizes a language L with $L \cap \{a^s b^s : s \ge s_0\} = \{a^{tn}b^{tn} : t = t_0, t_0 + 1, \ldots\}$ for some fixed $s_0, t_0 \in \mathbb{N}$.

5.4 Proofs of Theorem 7 and Proposition 8

Proof of Theorem 7. The lower bound can be obtained as a corollary of Lemma 12, by the following argument. Since $n|g_1g_2$, there exists some f such that $nf = g_1g_2$. Therefore, we can assume that $n = n_1n_2$ for some n_1, n_2 with $n_1 \mid g_1$ and $n_2 \mid g_2$. Furthermore, as $g_1, g_2 \le |Q|$, it also holds that $n_1, n_2 \le |Q|$, that is, $|Q| \ge \max\{n_1, n_2\}$. But $\max\{n_1, n_2\} \ge (n_1 + n_2)/2$, so it follows that $|Q| \ge (n_1+n_2)/2$, and so $|Q| \ge \min(n'_1+n'_2)/2$, with min over all factorizations $n = n'_1 n'_2$. This concludes the proof of the lower bound.

As mentioned above, we use the same idea as in Theorem 6 to prove the upper bound. Given a factorization $n = n_1n_2$, we show how to construct a VPA that accepts this language and satisfies the stated bounds. First note that $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\} = L_{\mathsf{balanced}} \cap L_{\mathsf{cycles}}$, where $L_{\mathsf{balanced}} = \{a^s b^s : s \ge 0\}$ and L_{cycles} is given by the definition in (2) on p. 347. It is straightforward to construct a VPA for L_{balanced} with 5 control states and 2 stack symbols, and a VPA for the intersection of languages is easily obtained by taking the products of control states and stack alphabets. Therefore, it suffices to show that L_{cycles} can be recognized with a VPA with $|Q| = n_1 + n_2 + O(1)$ and $|P| \le 3$, but we already know how to do this with Lemma 9. This completes the proof of Theorem 7. ◀

Proof of Proposition 8. The lower bound holds, because the intersection $L_n \cap \{a_1, b_1\}^*$ is essentially $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$, just with a_1, b_1 in place of a, b. As a result, the smallest VPA recognizing L_n can be at most a constant factor smaller than the smallest VPA recognizing $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$. As for the upper bound, it suffices to replace, in the VPA $\mathcal{V}(n_1, n_2)$ for L_{cycles}, transitions reading a and b with pairs of transitions reading a_i and b_j, and replace the VPA for L_{balanced} with a VPA for $\{a_{i_1} \ldots a_{i_k} b_{i_k} \ldots b_{i_1} : i_1, \ldots, i_k \in \{1, 2\}\}$. ◀

6 Open questions

Unanswered questions of a related nature abound. Probably the most natural is the following one: for which classes of machines are other modes of operation (nondeterministic, universal, and alternating ones) of help for the task of recognizing the language $\{1^n\}$? Kupferman et al. [12] provide a comprehensive answer for the classes of finite-state machines. For other classes, a short list of some specific problems follows:

1. Does nondeterminism or alternation make it possible for pushdown automata to count to n with $o(\log n)$ states for all n? The arguments that lead to the lower bound of $\Omega(\log n)$ in Theorem 3 only work for deterministic PDA and seem tricky or impossible to generalize. For other modes of operation, the only known lower bound is at most $\log n / \log\log n\,(1+o(1))$, given by the Kolmogorov argument of Corollary 2 and Remark 3.
2. Can nondeterministic or universal counter automata count to n with $o(\sqrt{n})$ states? As seen from the upper bound of Theorem 5, counter automata with unbounded alternation can, but is it achievable with bounded alternation depth?
3. What is the state complexity of recognizing the language $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$ with nondeterministic visibly pushdown automata?

Needless to say, many other related problem settings exist.

References

1 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–43, 2009. Revised version available at `http://robotics.upenn.edu/~alur/Jacm09.pdf`.

2 Jean-Camille Birget. Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory*, 29(3):191–226, 1996.

3 Can one-way alternating automata with one-counter recognize some unary non-regular languages? `http://cstheory.stackexchange.com/q/19046/13649`, 2013–2014.

4 Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13(4):547–569, 1966.

5 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

6 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and abhi shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

7 Dmitry Chistikov and Rupak Majumdar. Unary pushdown automata and straight-line programs. In *ICALP'14, Part II*, volume 8573 of *LNCS*, pages 146–157, 2014.

8 Marek Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.

9 Yuval Filmus. Lower bounds for context-free grammars. *Inf. Process. Lett.*, 111(18):895–898, 2011.

10 Matthew M. Geller, Harry B. Hunt III, Thomas G. Szymanski, and Jeffrey D. Ullman. Economy of description by parsers, DPDA's, and PDA's. *Theor. Comput. Sci.*, 4(2):143–153, 1977.

11 Seymour Ginsburg and H. Gordon Rice. Two families of languages related to ALGOL. *J. ACM*, 9(3):350–371, 1962.

12 Orna Kupferman, Amnon Ta-Shma, and Moshe Y. Vardi. Concurrency counts. Technical report, available at `http://www.cs.tau.ac.il/~amnon/Papers/KTV.submitted.cjtcs.ps`, 2001.

13 Ernst L. Leiss. Succinct representation of regular languages by boolean automata. *Theor. Comput. Sci.*, 13(3):323–330, 1981.

14 Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Texts and monographs in computer science. Springer, 1993.

15 Markus Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

16 Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In *SWAT (FOCS) 1971*, pages 188–191, 1971.

17 Alexander Okhotin, Xiaoxue Piao, and Kai Salomaa. Descriptional complexity of input-driven pushdown automata. In *Dassow Festschrift 2012*, volume 7300 of *LNCS*, pages 186–206, 2012.

18 Alexander Okhotin and Kai Salomaa. Complexity of input-driven pushdown automata. *SIGACT News*, 45(2):47–67, 2014.

19 Giovanni Pighizzini. Deterministic pushdown automata and unary languages. *Int. J. Found. Comput. Sci.*, 20(4):629–645, 2009.

20 Giovanni Pighizzini, Jeffrey Shallit, and Ming-wei Wang. Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. *J. Comput. Syst. Sci.*, 65(2):393–414, 2002.

21 Wojciech Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In *ICALP'04*, volume 3142 of *LNCS*, pages 15–27, 2004.

22 Kai Salomaa. Limitations of lower bound methods for deterministic nested word automata. *Inf. Comput.*, 209(3):580–589, 2011.

Mixed Nash Equilibria in Concurrent Terminal-Reward Games

Patricia Bouyer, Nicolas Markey, and Daniel Stan

LSV – CNRS & ENS Cachan – France

Abstract

We study mixed-strategy Nash equilibria in multiplayer deterministic concurrent games played on graphs, with terminal-reward payoffs (that is, absorbing states with a value for each player). We show undecidability of the existence of a constrained Nash equilibrium (the constraint requiring that one player should have maximal payoff), with only three players and 0/1-rewards (*i.e.*, reachability objectives). This has to be compared with the undecidability result by Ummels and Wojtczak for turn-based games which requires 14 players and general rewards. Our proof has various interesting consequences: (*i*) the undecidability of the existence of a Nash equilibrium with a constraint on the social welfare; (*ii*) the undecidability of the existence of an (unconstrained) Nash equilibrium in concurrent games with terminal-reward payoffs.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, D.2.4 Software/Program Verification), G.3 Probability and statistics

Keywords and phrases concurrent games, randomized strategy, Nash equilibria, undecidability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.351

1 Introduction

Games (especially games played on graphs) have been intensively used in computer science as a powerful way of modelling interactions between several computerised systems [11, 8]. Until recently, more focus had been put on the study of purely antagonistic games (a.k.a. zero-sum games, where the aim of one player is to prevent the other player from achieving her objective), which conveniently model systems evolving in a (hostile) environment.

Over the last ten years, games with non-zero-sum objectives have come into the picture: they allow for conveniently modelling complex infrastructures where each individual system tries to fulfill its own objectives, while still being subject to actions of the surrounding systems. As an example, consider (a simplified version of) the team formation problem [4], an example of which is presented in Fig. 1: several agents are trying to achieve tasks; each task requires some resources, which are shared by the players. Achieving a task thus requires the formation of a team that have all required resources for that task: each player selects the task she wants to achieve (and so proposes her resources for achieving that task), and if a task receives enough resources, the associated team receives the corresponding payoff (to be divided among the players in the team). In such a game, there is a need of cooperation (to gather enough resources), and an incentive to selfishness (to maximise the payoff).

In that setting, focusing only on optimal strategies for one single agent is not relevant. In game theory, several solution concepts have been defined, which more accurately represents rational behaviours of these multi-player systems; Nash equilibrium [9] is the most prominent such concept: a Nash equilibrium is a strategy profile (that is, one strategy to each player) where no player can improve her own payoff by unilaterally changing her strategy. In other terms, in a Nash equilibrium, each individual player has a satisfactory strategy with regards to the other players' strategies. Notice that Nash equilibria need not exist or be unique,

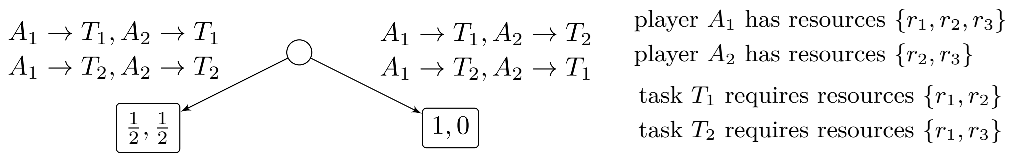

$A_1 \to T_1, A_2 \to T_1$
$A_1 \to T_2, A_2 \to T_2$
$A_1 \to T_1, A_2 \to T_2$
$A_1 \to T_2, A_2 \to T_1$
$\frac{1}{2}, \frac{1}{2}$
$1, 0$
player A_1 has resources $\{r_1, r_2, r_3\}$
player A_2 has resources $\{r_2, r_3\}$
task T_1 requires resources $\{r_1, r_2\}$
task T_2 requires resources $\{r_1, r_3\}$

Figure 1 An instance of the team-formation problem. For any deterministic choice of actions, one of the players has an incentive to change her choice: there is no pure Nash equilibrium. However there is one mixed Nash equilibrium, where each player plays T_1 and T_2 uniformly at random.

and are not necessarily optimal: Nash equilibria where all players lose may coexist with more interesting Nash equilibria. Therefore, looking for constrained Nash equilibria (*e.g.* equilibria in which some players are required to win, or equilibria with maximal social welfare) is an interesting and important problem to study, which has been suggested both in the game-theory community [5] and in the computer-science community [12].

In this paper, we study (deterministic) concurrent games played on graphs. Such games are indeed a general and relevant model for interactive systems, where the agents take their decision simultaneously (which is the case for instance in distributed systems). Concurrent games subsume turn-based games, where in each state, only one player has the decision for the next move, and which have attracted more focus until now in the computer science community. Notice also that in game theory, models are almost exclusively based on concurrent actions (*e.g.* games in normal form given as matrices indicating the payoff of each player for each concurrent choice of actions, and extensions thereof, such as repeated games).

In this paper we are interested in randomized (a.k.a. mixed) strategies for the players. A mixed strategy consists in choosing, at each step of the game, a probability distribution over the set of available actions; the game then proceeds following the product distribution of the strategies of all players. Strategies may depend on the history of the game, *i.e.*, the sequence of visited states, but do not require to see the actions played by the other agents. In previous works, the first two authors have focused on pure strategies, where at each step, each player proposes exactly one action, and developed algorithms for deciding the existence of constrained Nash equilibria in various settings [1]. In the present paper, we focus on terminal-reward payoffs (where some designated states are absorbing, and each player has a value—or reward—attached to each of these states): the payoff of a player is then her expected reward. We will also consider the subclass of games with terminal-reachability objectives, where the reward in each absorbing state is either 0 or 1 (hence the expected reward for a player is the probability to reach her winning states). The game in Fig. 1 has terminal-reward payoffs: they are given by the values labelling the two absorbing states (1 for player A_1 and 0 for player A_2 in the right-most state). This game can be shown to have no pure-strategy Nash equilibria, but it has a mixed-strategy one.

Our results. Our main result is the undecidability of the existence of a 0-optimal Nash equilibrium in concurrent games with terminal-reachability payoff functions, with only three players and strategies insensitive to actions. A 0-optimal Nash equilibrium is a Nash equilibrium in which one designated player is required to have maximal payoff (that is, 1 in the case of terminal-reachability payoffs). A corollary of our result is the undecidability of the existence of unconstrained Nash equilibria in concurrent games with terminal payoffs. We believe that these results are important, as they solve natural questions for basic objectives. Moreover, our constructions give new insight in the understanding of concurrent games and their algorithmics, and contain several intermediary tools that can be interesting on their own in different contexts.

Several results already exist in related settings:

- our result should first be compared with the undecidability of the existence of a 0-optimal Nash equilibrium in turn-based games with terminal-reward payoffs[13], which requires 14 players and general rewards. It should be noticed that this result requires more than 0/1 rewards (contrary to our result), since the existence of a 0-optimal equilibrium can be decided in polynomial time in turn-based games with terminal-reachability payoffs (by combining the reduction to pure 0-optimal Nash equilibria of [12] and the algorithm in [13] for computing such equilibria);
- our result should also be compared with polynomial-time algorithm for deciding the existence of a 0-optimal *pure* Nash equilibrium in concurrent terminal-reward games [15];
- our result has several corollaries, that we develop at the end of the paper:
  - the existence of a (unconstrained) Nash equilibrium in terminal-reward games with three players; on the opposite, stationary ϵ-Nash equilibria do always exist in concurrent games for terminal-reachability (and terminal-reward) games [3];
  - the existence of a Nash equilibrium that maximizes the social welfare in games with terminal-reachability payoffs is undecidable with three players. This should be compared with the NP-completeness of the existence of such equilibria for two-player normal-form games [6];
  - the existence of a constrained finite-memory Nash equilibrium in terminal-reachability games is undecidable with three players;
  - the existence of a constrained Nash equilibrium in safety games is undecidable with three players. This can be compared to the result of [10], which states that there always exists a Nash equilibrium (with little memory) in a safety game.

By lack of space, only some sketches of proofs could be included in this paper. We refer the interested reader to [2] for full details.

2 Definitions

▶ **Definition 1.** A concurrent arena $\mathcal{A}$ is a tuple $\mathcal{A} = \langle \mathsf{States}, \mathsf{Agt}, \mathsf{Act}, \mathsf{Tab}, (\mathsf{Allow}_i)_{i\in\mathsf{Agt}}\rangle$ where States is a finite set of states; Agt is a finite set of players; Act is a finite set of actions; For all $i \in \mathsf{Agt}$, $\mathsf{Allow}_i \colon \mathsf{States} \longrightarrow 2^{\mathsf{Act}}\backslash\{\varnothing\}$ is a function describing authorized actions in a given state for Player i; $\mathsf{Tab} \colon \mathsf{States} \times \mathsf{Act}^{\mathsf{Agt}} \to \mathsf{States}$ is the transition function.

A state $s \in \mathsf{States}$ is said *terminal* (or *final*) if $\mathsf{Tab}(s, \cdot) \equiv s$. We write $\mathsf{F}_{\mathcal{A}}$ (or simply F when the underlying arena is clear from the context) for the set of terminal states of $\mathcal{A}$.

A *history* of such an arena $\mathcal{A}$ is a finite, non-empty word $h \in \mathsf{States}^+$. We denote by $\mathsf{first}(h)$ and $\mathsf{last}(h)$ respectively the first and last states of the word h. During a play, players in Agt choose their next moves concurrently and independently from each others, according to the current history h and what they are allowed to do in the current state $\mathsf{last}(h)$.

▶ **Definition 2.** A *strategy* for Player i is a function $\sigma_i \colon \mathsf{States}^+ \to \mathsf{Dist}(\mathsf{Act})$ with the requirement that $\sigma_i(h)^{-1}(\mathbb{R}_{>0}) \subseteq \mathsf{Allow}_i(\mathsf{last}(h))$ for all history h.

Let $\alpha \in \mathsf{Act}$. We write $\sigma_i(\alpha \mid h)$ for the probability mass $\sigma_i(h)(\alpha)$ of action α in the distribution $\sigma_i(h)$. In the sequel, we sometimes write $\sigma_i(h) = \alpha$ when $\sigma_i(\alpha \mid h) = 1$. When $\sigma_i(h) \in \mathsf{Act}$ for all h, the strategy σ_i for player i is said to be *pure*. Otherwise it is said to be *mixed*. We denote by S_i (resp. $\mathbb{S}_i$) the set of pure (resp. mixed) strategies of Player i. A *strategy profile* σ is a mapping assigning one strategy to each player. We write $\mathbb{S}$ for the set of all strategy profiles, and for $\sigma \in \mathbb{S}$, we will write σ_i in place of $\sigma(i)$ for the strategy of Player i.

▶ Remark. While strategies are aware of the sequence of actions played in a turn-based game, we can notice this is generally not the case in the concurrent setting depicted here, since strategies only depend on the sequence of visited states. This is realistic when considering multi-agent systems, where only the global effect of the actions of the players is assumed to be observable. However this partial-information hypothesis makes the detection of strategy deviations (and therefore the computation of Nash equilibria) harder.

Consider a strategy profile $\sigma \in \mathbb{S}$ and an initial state s_0. For any history $h \in \mathsf{States}^+$ and any player $i \in \mathsf{Agt}$, we construct the random variable $\alpha_i(h) \in \mathsf{Act}$ with distribution $\sigma_i(h)$ such that $(\alpha_i(h))_{i\in\mathsf{Agt},h\in\mathsf{States}^+}$ is a family of independent random variables.

We define the stochastic process $(X_n)_{n\in\mathbb{N}}$ inductively by $X_0 = s_0$ and for every n, $X_{n+1} = X_n \cdot \mathsf{Tab}\left(\mathsf{last}(X_n), (\alpha_i(X_n))_i\right)$. For each n, the random variable X_n takes value in States^{n+1}: $(X_n)_n$ is an increasing sequence of prefixes whose limit is an infinite random run $X_\infty \in \mathsf{States}^\omega$.

We now consider the standard Borel σ-algebra over States^ω from s_0, and define the probability measure $\mathbb{P}^\sigma$ as the probability distribution induced by X_∞, that is, if B is a Borel subset of States^ω, $\mathbb{P}^\sigma(B) = \mathbb{P}(X_\infty \in B)$. It coincides with the standard construction based on cylinders. In the following, to make explicit the initial state, we may write $\mathbb{P}^\sigma(B \mid s_0)$ instead of simply $\mathbb{P}^\sigma(B)$. In the sequel, we sometimes also abusively write h for the cylinder $h \cdot \mathsf{States}^\omega$: then, when we write $\mathbb{P}^\sigma(h \mid s_0)$, we mean $\mathbb{P}^\sigma(X_{|h|} = h)$. If $\mathbb{P}^\sigma(h \mid s_0) > 0$, we say that σ enables h from s_0: in that case we can define the conditional probability $\mathbb{P}^\sigma(B \mid h) = \mathbb{P}^\sigma(B \mid X_{|h|} = h)$.

Finally we say that a node n is activated by a strategy profile whenever it is visited with positive probability under that profile.

▶ **Definition 3.** A *terminal-reward game* $\mathcal{G} = \langle \mathcal{A}, s, (\phi_i)_{i\in\mathsf{Agt}} \rangle$ is given by an arena $\mathcal{A}$, an initial state s, and for every player $i \in \mathsf{Agt}$, a real-valued function ϕ_i ranging over terminal states of $\mathcal{A}$. In the following, we extend ϕ_i to every $r \in \mathsf{States}^\omega$, by $\phi_i(r) = \phi_i(s)$ if r is an infinite path ending in a state $s \in \mathsf{F}$, and $\phi_i(r) = 0$ otherwise.

The game $\mathcal{G}$ will be said a *terminal-reachability game* whenever each function ϕ_i only takes values 0 or 1.

▶ Remark. In the sequel, we represent terminal-reward games as graphs with circle states representing non-terminal states, and rectangle states representing terminal states, decorated with the associated rewards for all players. The self-loop on terminal states will be omitted. The transition table of the underlying arena is encoded by decorating the transitions with the move vectors that trigger it. Move vectors are written as words over Act, by identifying Agt with the subset $[\![0, |\mathsf{Agt}| - 1]\!]$. We will use $\cdot$ as a special symbol representing any action. Also, for a set S of words in $(\mathsf{Act} \cup \{\cdot\})^k$, with $k < |\mathsf{Agt}|$, and for a letter $a \in \mathsf{Act} \cup \{\cdot\}$, we write aS for the words $\{aw \mid w \in S\}$. See Fig. 2 (and the subsequent figures) for an example.

Consider a terminal-reward game $\mathcal{G}$, a strategy profile σ, and an enabled history h. One can easily check that ϕ_i is a mesurable function under $\mathbb{P}^\sigma$. The expected payoff of Player i under σ after h is defined as

$$\mathbb{E}^\sigma(\phi_i \mid h) = \sum_{x\in\mathsf{Img}(\phi_i)} x \cdot \mathbb{P}^\sigma\left(\phi_i^{-1}(\{x\}) \mid h\right).$$

In case $\mathcal{G}$ is a terminal-reachability game, the expected payoff of Player i is the probability of reaching terminal states with value 1 under ϕ_i.

Let $\mathcal{G}$ be a terminal-reward game. Let $\sigma \in \mathbb{S}$ be a (mixed) strategy profile in $\mathcal{G}$, and h be a history. A *single-player deviation* (simply called *deviation* hereafter, as we only consider deviations of a single player at a time) of σ for Player i after history h is another strategy profile σ' for which there exists $\sigma''_i \in \mathbb{S}_i$ satisfying

$$\forall h' \in \mathsf{States}^+ . \begin{cases} h \sqsubseteq h' \Rightarrow \sigma'_i(h') = \sigma''_i(h') \\ \forall j \in \mathsf{Agt}.\ (h \not\sqsubseteq h' \vee j \neq i) \Rightarrow \sigma'_j(h') = \sigma_j(h') \end{cases}$$

where $\sqsubseteq$ is the prefix relation. We then write $\sigma' = \sigma[i/\sigma''_i]^h$. The deviation $\sigma[i/\sigma''_i]^h$ is said *deterministic* if σ''_i is.

▶ **Definition 4.** Let $\mathcal{G}$ be a terminal-reward game. A strategy profile σ forms a *Nash equilibrium* after a history h when the following conditions are met:

- $h \in \mathsf{States}^+$ is enabled by σ from $\mathsf{first}(h)$;
- No player has a profitable deviation; in other terms, for all $i \in \mathsf{Agt}$ and for all $\sigma'_i \in \mathbb{S}_i$, it holds $\mathbb{E}^{\sigma[i/\sigma'_i]^h}(\phi_i \mid h) \leq \mathbb{E}^{\sigma}(\phi_i \mid h)$.

We then write that $\langle \sigma, h\rangle$ is a Nash equilibrium.

A Nash equilibrium $\langle \sigma, h\rangle$ is said 0-*optimal* whenever the expected payoff of Player 0 is optimal, that is, $\mathbb{E}^{\sigma}(\phi_i \mid h) = \max(\mathsf{Img}(\phi_0))$. In case of a terminal-reachability game, it amounts to saying that the payoff of Player 0 is 1.

The following result will be useful all along the paper:

▶ **Lemma 5.** *Let $\mathcal{G}$ be a terminal-reward game, and $\langle \sigma, h\rangle$ be a Nash equilibrium. If $\langle \sigma, h\rangle$ enables h', then $\langle \sigma, h'\rangle$ is a Nash equilibrium.*

In general, several Nash equilibria may coexist. It is therefore very relevant to look for *constrained Nash equilibria*, that is, Nash equilibria that satisfy a constraint on the expected payoff. In this paper, we only consider 0-optimality as the constraint, and we prove that the existence of a 0-optimal Nash equilibrium in a three-player terminal-reachability game is undecidable. To prove this result, we will first show undecidability in the case of terminal-reward games, and then extend the result to terminal-reachability games. Those results will have interesting corollaries, like the undecidability of the existence of a Nash equilibrium (with no constraint) in terminal-reward games, when the rewards are in $\{-1, 0, 1\}$, or the existence of a Nash equilibrium with optimal social welfare.

3 Tools

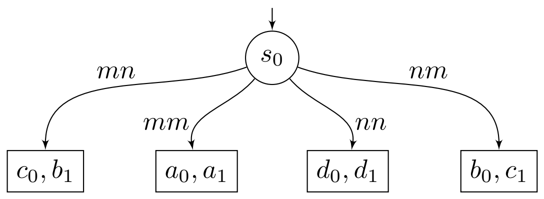
In this section, we develop several intermediary results that will be useful for our reduction. We first show that we can equivalently define Nash equilibria by considering only deterministic deviations (for non-negative terminal-reward games). We then study a few simple games and constructions which will be used in the encoding.

3.1 Deterministic deviations

We explain in this section that it is enough to consider deterministic deviations in the characterization of a Nash equilibrium.

▶ **Proposition 6.** *Let $\mathcal{G}$ be a terminal-reward game with non-negative rewards. Pick a history $h \in \mathsf{States}^+$, and a strategy profile σ. Then $\langle \sigma, h\rangle$ is a Nash equilibrium if, and only if, for all $i \in \mathsf{Agt}$ and all* deterministic *deviation $\sigma''_i \in S_i$, it holds $\mathbb{E}^{\sigma[i/\sigma''_i]^h}(\phi_A \mid h) \leq \mathbb{E}^{\sigma}(\phi_i \mid h)$.*

▶ Remark. A similar result was proven in [15, Proposition 3.1] for turn-based games with qualitative Borel objectives (the payoff is 1 if the run belongs to the designed objective, and it is 0 otherwise).



(a) A generic two-player two-action one-stage game

1 \ 0	m	n
m	a_0, a_1	b_0, c_1
n	c_0, b_1	d_0, d_1

(b) Associated matrix representation

Figure 2 Representations of a one-shot game.

3.2 One-stage games

We analyse two-player two-action one-stage games (that is, games that end up in a terminal state in one step), and obtain useful properties of their Nash equilibria. Such games can be represented by a graph as shown in Fig. 2a. Alternatively, these games, also known as one-shot games, can be represented as a matrix as in Table 2b (this is the standard representation in the game-theory community).

▶ **Lemma 7.** *Consider the two-player two-action one-shot concurrent game $\mathcal{G}$ of Fig. 2, and pick some strategy profile σ. If $\langle \sigma, s_0 \rangle$ is a Nash equilibrium, then for every player $i \in \{0, 1\}$, it holds*

$$\begin{cases} \sigma_i(m \mid s_0) < 1 & \Rightarrow \quad [(d_i - c_i) + (a_i - b_i)] \cdot \sigma_{1-i}(m \mid s_0) \leq d_i - c_i \\ \sigma_i(m \mid s_0) > 0 & \Rightarrow \quad [(d_i - c_i) + (a_i - b_i)] \cdot \sigma_{1-i}(m \mid s_0) \geq d_i - c_i \end{cases}$$

3.3 k-action matching-pennies games

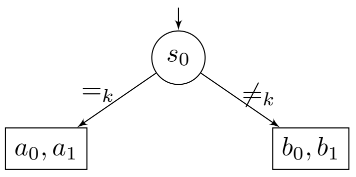


Figure 3 Matching-pennies game.

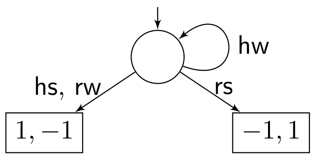
The classical matching-pennies games are a special case of one-stage games, where $a_i = d_i$ and $b_i = c_i$: basically, there are two outcomes, depending on whether the players propose the same action or not. This game can be generalized to k (≥ 2) actions, as depicted on Fig. 3. In this figure (and in the sequel), $=_k$ (resp. $\neq_k$) is a shorthand for pairs of identical (resp. different) actions taken from a set of k actions $\Sigma_k = \{c_1, \ldots, c_k\}$. In other terms, $=_k$ represents the set of words $\{c_i c_i \mid 1 \leq i \leq k\}$, and $\neq_k$ is the complement in Σ_k^2.

▶ **Lemma 8.** *In the k-action matching-pennies game, playing uniformly at random for both players defines a Nash equilibrium. Moreover, this is the unique Nash equilibrium of the game if, and only if, either $a_0 < b_0$ and $a_1 > b_1$, or $a_0 > b_0$ and $a_1 < b_1$. The payoff of this Nash equilibrium is $\left(\frac{1}{k} \cdot a_0 + \left(1 - \frac{1}{k}\right) \cdot b_0, \frac{1}{k} \cdot a_1 + \left(1 - \frac{1}{k}\right) \cdot b_1\right)$.*

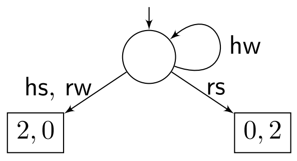
3.4 Games without equilibrium

In this section, we show that there are games that admit no Nash equilibria. We then explain how these games can be used to impose constraints on payoffs.

Consider the game *hide-or-run*, depicted in Fig. 4a. Player 0 can either hide (h) or run home (r), while Player 1 can either shoot him (s), or wait (w). If Player 1 shoots while Player 0 is hiding, she loses her bullet and loses the game. If Player 1 shoots when Player 0



(a) $\mathcal{H}$ has no Nash equilibria



(b) $\mathcal{H}'$ does have a Nash equilibrium

Figure 4 Hide-or-run games.

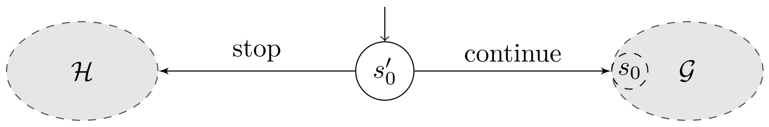


Figure 5 A game that has a Nash equilibrium if, and only if, $\mathcal{G}$ has a 0-optimal Nash equilibrium.

is running, she wins. This game has been shown to have no optimal almost-sure strategy [7], and we adapt the proof to show that it has no Nash equilibria.

▶ **Lemma 9.** *The game $\mathcal{H}$ has no Nash equilibria.*

The payoff function of $\mathcal{H}$ takes negative value. In order to only have nonnegative payoffs, we could shift the values by 1, which yields the game $\mathcal{H}'$ depicted on Fig. 4b. But then one easily sees that the strategies $\sigma_0(\mathsf{h} \mid s_0^n) = 1$ and $\sigma_1(\mathsf{s} \mid s_0^n) = 1$ form a Nash equilibrium, contrary to a claim in [3, 13]. The difference is that when shifting the payoffs, we did not modify the payoff of the run that never reaches a terminal state: while this run was a positive deviation for Player 1 in $\mathcal{H}$, this is not the case in $\mathcal{H}'$ anymore.

We now explain how we use the game $\mathcal{H}$ to impose a 0-optimality constraint on the payoff. In the sequel, we restrict[1] to games where $\max_{s \in \mathsf{F}} \phi_0(s) = 1$. Then:

▶ **Lemma 10.** *Let $\mathcal{G}$ be a terminal-reward game. Then we can build a terminal-reward game $\mathcal{G}'$ (see Fig. 5) such that $\mathcal{G}$ has a 0-optimal Nash equilibrium if, and only if, $\mathcal{G}'$ has a Nash equilibrium.*

This lemma will be useful for extending the undecidability result from the constrained existence to the existence problem (Corollary 14).

▶ Remark. Note that in the above construction, game $\mathcal{H}$ can be replaced by any game with no Nash equilibria, such that Player 0 can secure a payoff $1 - \varepsilon$ for every $\varepsilon > 0$. For instance, one could use a game with limit-average payoff and nonnegative rewards only [14].

4 Updating values

Our undecidability proof will be based on an encoding of a two-counter machine. In this section and in the next one, we present games that will be building blocks for our proof.

Consider the game $\mathcal{G}_k^r$ depicted on Fig. 6: in this game, Player 0 has two available actions a and b from s_0, s_k and s_l, while the other two players can either continue (action c), or

[1] This is no loss of generality, since we can linearly rescale payoffs of each player without changing profitable deviations.

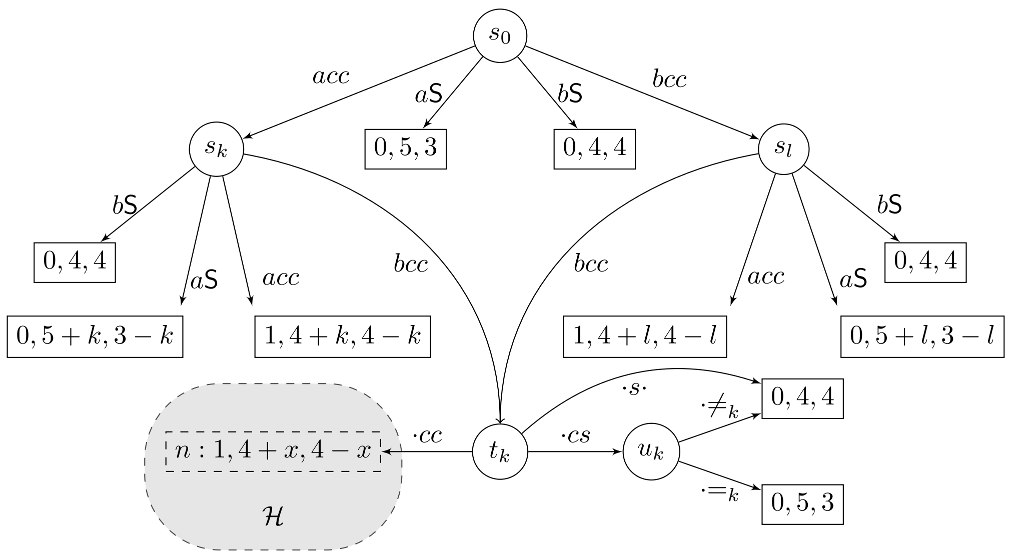


Figure 6 The rescale game $\mathcal{G}_k^r$ for $k \in \{1,2,3\}$ and $l = k-1$.

unilateraly decide to stop the game (action s) and go to a terminal state (where Player 0 will have payoff 0). In node t_k, only players 1 and 2 have a choice: they can either continue to the game $\mathcal{H}$ (when both of them play c), or decide to stop and go to a k-action matching-pennies game (when one of them plays s). In Fig. 6, we write S as a shorthand representing any combination of moves of players 1 and 2 where at least one of them decides to stop (action s). Node n is the initial node of a game $\mathcal{H}$ (which is unknown for the moment).

The interesting property of game $\mathcal{G}_k^r$ is that we can relate 0-optimal Nash equilibria from s_0 and those from n: (roughly) there is a Nash equilibrium from n of expected payoff $(1, 4+x, 4-x)$ if, and only if, there is a Nash equilibrium from s_0 of expected payoff $(1, 4+k\cdot x, 4-k\cdot x)$.

This is because, from s_0 and s_k, there is a threat for Player 0 that one of the players 1 and 2 stops the game immediately, leading to a state with payoff 0 for her. Hence, Player 0 is forced to "collaborate" with players 1 and 2 and help them be satisfied with their payoffs, either by joining one of the interesting terminal states of $\mathcal{G}_k^r$, or in the next game $\mathcal{H}$ after n. Some technical calculations show that Player 0 has to play a with probability $k \cdot x$ at s_0, and with probability $x/(x+1)$ at s_k and s_l. The gadget to the right of t_k is just for ensuring that $0 \leq k \cdot x \leq 1$ (this condition is required for having the above-mentioned equivalence between Nash equilibria from s_0 and Nash equilibria from n).

5 Comparing values

5.1 Testing game

We present in this section the construction of a game for comparing the expected payoffs in different nodes. This will be useful in our reduction to encode the zero-tests of our two-counter machine.

Consider the game $\mathcal{G}_t$ depicted on Fig. 7. This game has the very interesting property that if we assume there are 0-optimal Nash equilibria from n_1 and n_2 of respective payoffs $(1, 4+x, 4-x)$ and $(1, 4-y, 4+y)$, then there is a 0-optimal Nash equilibrium from s_0 if, and only if, $x = y$, and the payoff is then $(1, 4+x/2, 4-x/2)$. Indeed, unless $x = 0$ or $y = 0$,

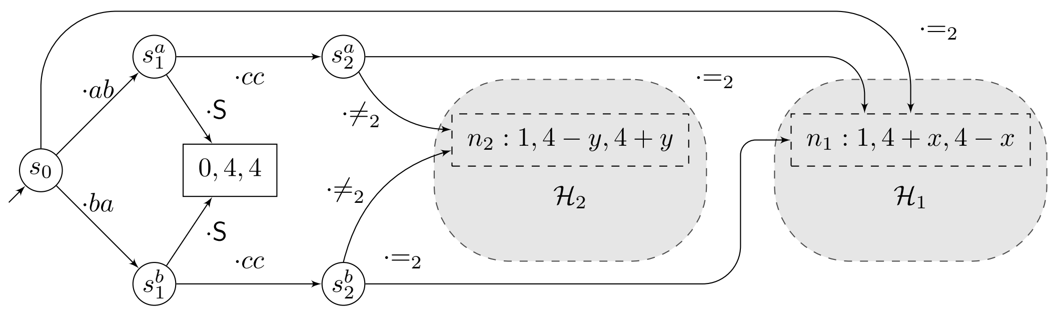


Figure 7 The testing game $\mathcal{G}_t$.

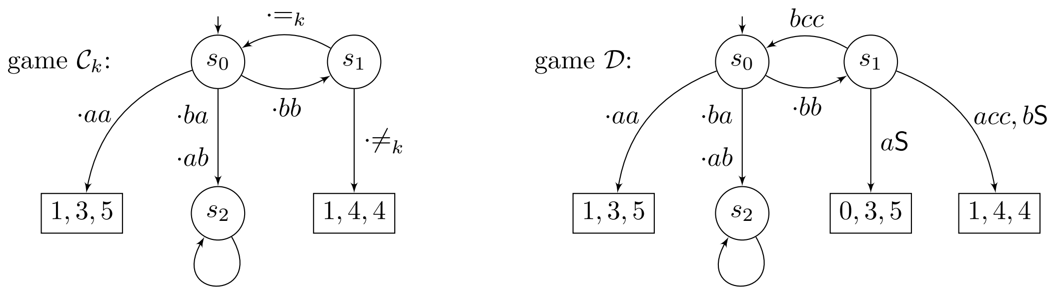


Figure 8 The modules $\mathcal{C}_k$ (for $k \geq 2$) and $\mathcal{D}$. Notice that state s_2 should be considered terminal, as it only carries a self-loop. We could replace it by a two-state loop. We could also see it as a terminal state with reward $(0,0,0)$, but for the proof of Corollary 16, we want the terminal rewards of players 1 and 2 to always sum to 8, which we could not achieve easily in this case.

it should be the case that a 0-optimal Nash equilibrium activates all states s_j^α in the game, and then, as players 1 and 2 have zero-sum objectives, the best way is to play uniformly at random in all states where this makes sense (when actions a and b are available), and to play deterministically action c in all states where c is available.

This gadget allows, by plugging in n_2 a game with known payoffs (the games on the next subsection), to check that the payoff at s_0 has some particular value (which depends on that after n_1).

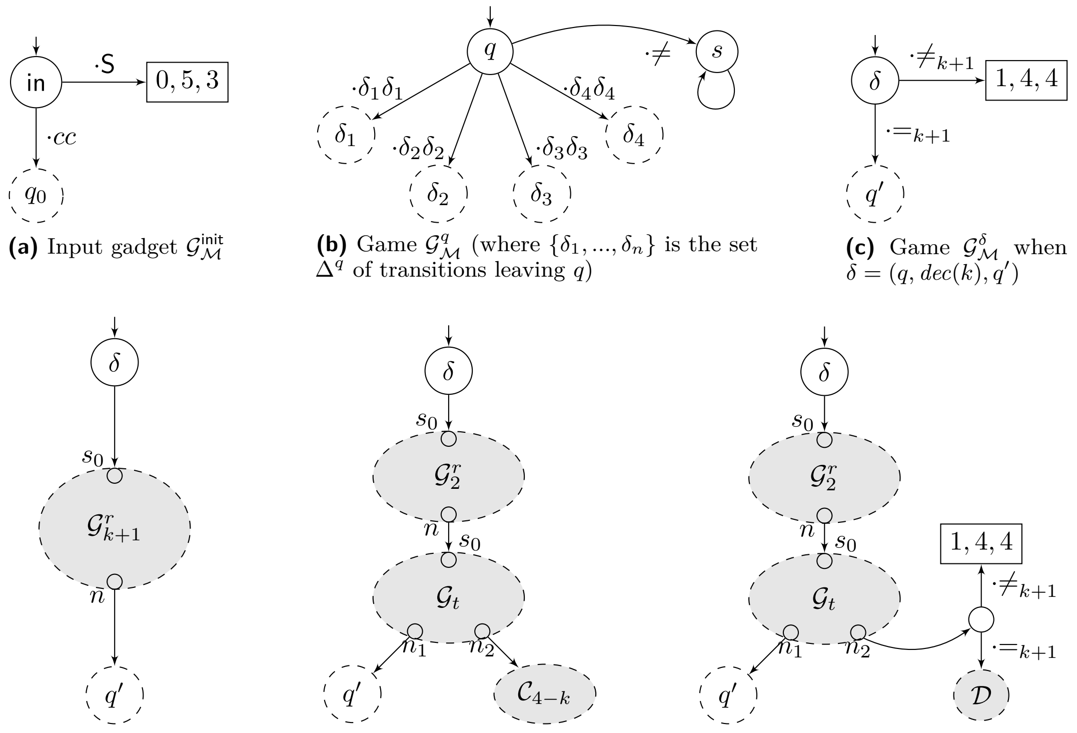
5.2 Counting modules

We now present games that generate a family of Nash equilibria with a particular expected payoffs. These modules will later be plugged at node n_2 of game $\mathcal{G}_t$, and will ensure that the payoff of an Nash equilibrium in $\mathcal{G}_t$ will have a predefined form.

▶ **Lemma 11.** *Consider the games of Fig. 8. For $n \in \mathbb{N} \cup \{+\infty\}$, define*

$$r_k(n) = \left(1, 4 - \frac{1}{k^n}, 4 + \frac{1}{k^n}\right) \qquad s(n) = \left(1, 4 - \frac{1}{n+1}, 4 + \frac{1}{n+1}\right)$$

Then the set of 0-optimal Nash equilibrium values is $\{s(n) \mid n \in \mathbb{N} \cup \{\infty\}\}$ in $\mathcal{D}$, and $\{r_k(n) \mid n \in \mathbb{N} \cup \{\infty\}\}$ in $\mathcal{C}_k$ for all $k \geq 2$.



(a) Input gadget $\mathcal{G}^{\mathsf{init}}_{\mathcal{M}}$

(b) Game $\mathcal{G}^q_{\mathcal{M}}$ (where $\{\delta_1, ..., \delta_n\}$ is the set Δ^q of transitions leaving q)

(c) Game $\mathcal{G}^\delta_{\mathcal{M}}$ when $\delta = (q, dec(k), q')$

(d) Game $\mathcal{G}^\delta_{\mathcal{M}}$ when $\delta = (q, inc(k), q')$

(e) Game $\mathcal{G}^\delta_{\mathcal{M}}$ when $\delta = (q, zero(k), q')$

(f) Game $\mathcal{G}^\delta_{\mathcal{M}}$ when $\delta = (q, !zero(k), q')$

Figure 9 Description of the subgames $\mathcal{G}^q_{\mathcal{M}}$ and $\mathcal{G}^\delta_{\mathcal{M}}$.

6 Undecidability proof

We now turn to the global undecidability proof of the constrained-existence problem in three-player games. The proof is a reduction from the recurring problem of a two-counter machine. We encode the behaviour of a two-counter machine $\mathcal{M}$ as a concurrent game $\mathcal{G}_{\mathcal{M}}$, which connects the various subgames depicted on Figures 9 (one initial gadget, one per state q, one per transition δ). Roughly, this game will encode a configuration (q, c_1, c_2) of $\mathcal{M}$ using a Nash equilibrium $\sigma \in \mathbb{S}$ from q such that $\mathbb{E}^\sigma(\phi \mid s) = \left(1, 4 + \frac{1}{2^{c_1}3^{c_2}}, 4 - \frac{1}{2^{c_1}3^{c_2}}\right)$ (property $P(q, c_1, c_2)$). Using the various constructions we have made previously, we can show that if $P(q, c_1, c_2)$ is satisfied, then there is a transition $(q, c_1, c_2) \rightarrow_\delta (q', c_1', c_2')$ in $\mathcal{M}$ such that $P(q', c_1', c_2')$ is satisfied as well, which allows to progress 'along' a Nash equilibrium while building a computation in $\mathcal{M}$.

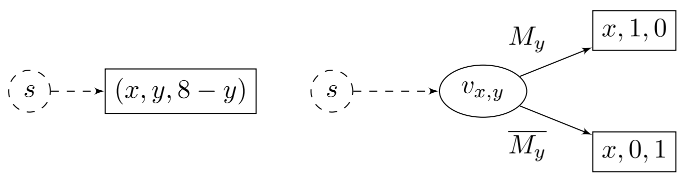
The correspondence between $\mathcal{M}$ and $\mathcal{G}_{\mathcal{M}}$ is made precise as follows:

▶ **Proposition 12.** *The two-counter machine $\mathcal{M}$ has an infinite valid computation if, and only if, there is a* 0*-optimal Nash equilibrium from state* in *in game $\mathcal{G}_{\mathcal{M}}$.*

This immediately entails:

▶ **Theorem 13.** *We cannot decide whether there exists a* 0*-optimal Nash equilibrium in three-player games with non-negative terminal-reward payoffs.*

We now consider several extensions of this result. We first state two straightforward corollaries. First, applying Lemma 10, we can enforce the 0-optimality constraint in the game by inserting an initial module. It follows:



	0	1	2	3	4	5	6	7
0	▓	▓	▓	▒	░			
1		▓	▓	▓	▒	░		
2			▓	▓	▓	▒	░	
3				▓	▓	▓	▒	░
4	░				▓	▓	▓	▒
5	▒	░				▓	▓	▓
6	▓	▒	░				▓	▓
7	▓	▓	▒	░				▓

▓ $=M_3$ ▒ $=M_4$ ░ $=M_5$

Figure 10 Transformation of a terminal node $(x, y, 8 - y)$ with an intermediate node $v_{x,y}$. The table on the right gives the value of M_y for some values of y (notice that $M_y \subseteq M_{y'}$ when $y \leq y'$).

▶ **Corollary 14.** *We cannot decide whether there exists a Nash equilibrium in three-player games with (possibly negative) terminal-reward payoffs.*

Now we realize that in this reduction, there is a 0-optimal Nash equilibrium from in if, and only if, there is a Nash equilibrium with social welfare larger than or equal to 9, where the social welfare is defined as the sum of the expected payoffs of all players. As an immediate corollary, we get:

▶ **Corollary 15.** *We cannot decide whether there exists a Nash equilibrium with some lower bound on the social welfare (or with optimal social welfare) in three-player terminal-reward games with non-negative payoffs.*

We now explain briefly how the main theorem can be extended to terminal-reachability payoffs. We indeed realize that the payoffs of players 1 and 2 always sum up to 8 in the reduction (the game between those two players is zero-sum). The idea is then to replace each terminal state with a simple gadget in which the payoffs of players 1 and 2 are $(8, 0)$ or $(0, 8)$, and to use an adequate set of actions which decomposes runs into two sets with proportions mimicking the normal rewards of the terminal state. For instance, for a reward $(x, y, 8 - y)$, the set of actions $M_y = \{\cdot ij \mid \exists 0 \leq r < y.\ i - j = r \mod 8\}$ will lead to $(x, 8, 0)$ and its complement to $(x, 0, 8)$, as illustrated on Fig. 10. In the game from $v_{x,y}$, there is a unique Nash equilibrium which consists in playing uniformly at random for both players, yielding a payoff of $(x, y, 8 - y)$. It remains to normalize and replace each $(x, 8, 0)$ (resp. $(x, 0, 8)$) by $(x, 1, 0)$ (resp. $(x, 0, 1)$).

▶ **Corollary 16.** *We cannot decide whether there exists a* 0*-optimal Nash equilibrium in three-player games with terminal-reachability payoffs.*

Finally, (roughly) by dualizing reachability and safety conditions, we can prove that the constrained existence of Nash equilibrium in safety games cannot be decided. This is to be compared with the fact that there always exists a Nash equilibrium in safety games [10].

▶ **Corollary 17.** *We cannot decide whether there exists a Nash equilibrium in three-player safety games with payoff* 0 *assigned to Player* 0.

7 Conclusion and future work

In this paper we have shown the undecidability of the existence of a constrained Nash equilibrium in a three-player concurrent game with terminal-reachability objectives. We believe this result is surprising, since it applies to very simple payoff functions, and with very few players. This result has to be compared with the undecidability result of [13], which on one hand, applies to turn-based games, but requires 14 players and the full power of terminal-reward payoffs. Furthermore, in turn-based games with terminal-reachability payoffs, constrained Nash equilibria can be computed (in polynomial time) through a reduction to pure Nash equilibria [12] and algorithms for computing pure Nash equilibria [13]. We have also mentioned a couple of interesting corollaries that we do not repeat here.

This work lets open the decidability status of the constrained-existence problem in two-player games with terminal-reward and terminal-reachability payoffs. In fact, even the existence of Nash equilibria in such games is an open problem: it was believed until recently that there are two-player games with nonnegative terminal rewards having no Nash equilibrium [3, 13], but the proposed example was actually wrong (as we explained in Section 3.4). If one can find such a game with no Nash equilibrium, then our Corollary 14 extends to nonnegative terminal-reward games, and possibly to terminal-reachability games. Notice that two-player games have been studied quite a lot in the literature, and we know for instance that (uniform) ϵ-Nash equilibria always exist in terminal-reward games [16, 17].

References

1 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash equilibria in concurrent games. *Logical Methods in Computer Science*, 2014. Submitted, under revision.

2 Patricia Bouyer, Nicolas Markey, and Daniel Stan. Mixed Nash equilibria in concurrent terminal-reward games. Research Report LSV-14-11, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2014.

3 Krishnendu Chatterjee, Marcin Jurdziński, and Rupak Majumdar. On Nash equilibria in stochastic games. In *CSL'04*, volume 3210 of *LNCS*, pages 26–40. Springer, 2004.

4 Taolue Chen, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Verifying team formation protocols with probabilistic model checking. In *CLIMA XII*, volume 6814 of *LNAI*, pages 190–207. Springer, 2011.

5 Vincent Conitzer and Tuomas Sandholm. Complexity results about Nash equilibria. In *IJCAI'03*, pages 765–771, 2003.

6 Vincent Conitzer and Tuomas Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.

7 Luca de Alfaro, Thomas Henzinger, and Orna Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3):188–217, 2007.

8 Thomas A. Henzinger. Games in system design and verification. In *TARK'05*, pages 1–4, 2005.

9 John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, 1950.

10 Piercesare Secchi and William D. Sudderth. Stay-in-a-set games. *International Journal of Game Theory*, 30:479–490, 2001.

11 Wolfgang Thomas. Infinite games and verification. In *CAV'02*, volume 2404 of *LNCS*, pages 58–64. Springer, 2002. Invited Tutorial.

12 Michael Ummels. The complexity of Nash equilibria in infinite multiplayer games. In *FoSSaCS'08*, volume 4962 of *LNCS*, pages 20–34. Springer, 2008.

13 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in limit-average games. In *CONCUR'11*, volume 6901 of *LNCS*, pages 482–496. Springer, 2011.

14 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in limit-average games. Technical Report abs/1109.6220, CoRR, 2011.

15 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in stochastic multiplayer games. *Logical Methods in Computer Science*, 7(3), 2011.

16 Nicolas Vieille. Equilibrium in two-person stochastic games I: A reduction. *Israel Journal of Mathematics*, 119:55–91, 2000.

17 Nicolas Vieille. Equilibrium in two-person stochastic games II: The case of recursive games. *Israel Journal of Mathematics*, 119:93–126, 2000.

Quantitative Games with Interval Objectives*

Paul Hunter and Jean-François Raskin

Département d'Informatique, Université Libre de Bruxelles (U.L.B.)
`{paul.hunter,jraskin}@ulb.ac.be`

Abstract

Traditionally quantitative games such as mean-payoff games and discount sum games have two players – one trying to maximize the payoff, the other trying to minimize it. The associated decision problem, "Can Eve (the maximizer) achieve, for example, a positive payoff?" can be thought of as one player trying to attain a payoff in the interval $(0, \infty)$. In this paper we consider the more general problem of determining if a player can attain a payoff in a finite union of arbitrary intervals for various payoff functions (liminf/limsup, mean-payoff, discount sum, total sum). In particular this includes the interesting exact-value problem, "Can Eve achieve a payoff of exactly (e. g.) 0?"

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Quantitative games, Mean-payoff games, Discount sum games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.365

1 Introduction

Quantitative two-player games on graphs have been extensively studied in the verification community [8, 6, 14, 10, 18]. Those models target applications in reactive system synthesis with resource constraints. In these games two players, Eve and Adam, interact by moving a token around a weighted, directed graph, for a possibly infinite number of moves. This interaction results in a play which is an infinite path in the graph. The value of the play is computed by applying a payoff function to the sequence of weights of the edges traversed along the path. Typical payoff functions are (lim)sup, (lim)inf, mean-payoff, (total) sum, and discounted sum.

In the literature is usual to assume that Eve is attempting to maximize the payoff and Adam is attempting to minimize it. In this context all these games are determined, that is the maximum that Eve can ensure is equal to the minimum that Adam can ensure, and this value can be computed in polynomial time for (lim)inf and (lim)sup [5], and in pseudo-polynomial time for mean-payoff, discounted sum, and total sum [18, 10]. The associated decision problem is the *threshold problem*: Given a game graph, a payoff function and a threshold ν does Eve have a strategy to ensure all consistent plays have payoff at least ν? The threshold problems for the aforementioned payoff functions are all closely related, and it is known that Eve and Adam can play optimally in those games with *memoryless strategies* [11]. Consequently the decision problem for all those games is in $\mathsf{NP} \cap \mathsf{coNP}$. In fact, it can be shown in $\mathsf{UP} \cap \mathsf{coUP}$ for mean-payoff, discounted sum, and total sum, and in PTIME for (lim)inf and (lim)sup.

The threshold problem can be seen as game in which Eve is trying to force the payoff to belong to the interval of values $[\nu, \infty)$. In this paper we consider the more general problem of determining if a player can attain a payoff in a finite union of arbitrary intervals for

* This work was supported by the ERC inVEST (279499) project.

Table 1 Complexity of deciding the winner in interval games.

Payoff type	Single interval	Multiple intervals	
		Binary	Unary
Liminf/limsup	PTIME	NP ∩ coNP	Parity game-c
Mean-payoff	NP ∩ coNP	PSPACE	Parity game-hard
Discounted sum (non-singleton)	PSPACE-c		PTIME
Discounted sum (exact value)	PSPACE-hard		?
Total sum	EXPSPACE-c		PSPACE-c

Table 2 Memory requirements for interval games.

Payoff type	Single interval (Eve/Adam)	Multiple intervals
Liminf/limsup	Positional	
Mean-payoff	Finite/Positional	Infinite
Discounted sum (non-singleton)	Finite	
Discounted sum (exact value)	Infinite	
Total sum	Finite/Infinite	Infinite

the classical payoff functions mentioned above. That is, we are interested in the following question: Given a weighted arena G and a finite union of real intervals, what is the complexity of determining if Eve has a winning strategy to ensure the payoff of any consistent play lies within the interval union? In particular this includes the interesting exact-value problem: Can Eve achieve a payoff of exactly ν? Such objectives arise when considering efficiency constraints, for example can a system achieve a certain payoff without exceeding a certain target? We consider two versions of our problem depending on whether the numeric inputs (weights, interval bounds and discount factor) are given in binary or unary. We also consider the memory requirements for a winning strategy both for Eve and Adam. Our games are a natural subclass of multi-dimensional quantitative games (see e. g. [6]), however our results are largely incomparable with that paper as we consider a wider array of payoff functions and our objective corresponds to *disjunctions* of multi-dimensional objectives which were not considered.

Tables 1 and 2 summarize the results of this paper: the first table highlights the complexity results and the second table highlights the memory requirements for playing optimally. While the classical threshold problems for weighted games can be solved in PTIME for (lim)inf and (lim)sup and in NP ∩ coNP for mean-payoff, discounted sum and total sum, and memoryless strategies always suffice, the situation for our interval objectives is far richer:

- For liminf and limsup, we provide a polynomial time algorithm in the case of a single interval. For a union of intervals, we show that these games are polynomially equivalent to parity games: so we can solve them in NP ∩ coNP, and a polynomial time algorithm for interval liminf games would provide a polynomial time algorithm for parity games (a long-standing open question in the area). Optimal strategies are memoryless for both players.
- For interval mean-payoff games, we provide a recursive algorithm that executes in polynomial space. This algorithm leads to a NP ∩ coNP algorithm in the case of single interval objectives. While mean-payoff games can be solved in polynomial time when weights are given in unary, we show here that interval mean-payoff games are at least as

hard as parity games even when weights are given in unary. So, a pseudo-polynomial time algorithm for interval mean-payoff games would lead to a polynomial algorithm for parity games. For a union of intervals, infinite memory may be necessary for both players, and for single interval exponential memory may be necessary for Eve while Adam can always play a memoryless strategy.

- Interval discounted sum games are complete for polynomial space when singleton intervals (and singleton gaps between intervals) are forbidden. The decidability for the case when singletons are allowed is left open and it generalizes known open problems in single player discounted sum graphs [7, 1]. Finite memory suffices for both players in the non-singleton case and infinite memory is needed for both players when singletons are allowed.
- For the total sum payoff, we establish a strong link with one counter parity games that leads to a PSPACE-complete result for unary encoding and an EXPSPACE-complete result for binary encoding. For single interval games Eve need only play finite memory strategies, while she may need infinite memory in the general case. In both cases, Adam may require infinite memory.

Structure of the paper. Section 2 introduces the necessary preliminaries. In Sections 3, 4, 5, and 6 we consider the decision problems and memory requirements for the liminf/limsup, mean-payoff, discounted sum, and total sum payoff functions, respectively. Due to space restrictions the full details of several proofs are left for the full version.

2 Preliminaries

A game graph is a tuple $G = (V, V_\exists, E, w, q_0)$ where (V, E, w) is an edge-weighted graph, $V_\exists \subseteq V$, and $q_0 \in V$ is the initial state. Without loss of generality we will assume all weights are integers. In the sequel we will depict vertices in $V_\exists$ with squares and vertices in $V \setminus V_\exists$ with circles. In complexity analyses we will denote the maximum absolute value of a weight in a game graph by W. If $V' \subseteq V$, we denote by $G \setminus V'$ the game graph induced by $V \setminus V'$.

A play in a game graph is an infinite sequence of states $\pi = v_0v_1\cdots$ where $v_0 = q_0$ and $(v_i, v_{i+1}) \in E$ for all i. Given a play $\pi = v_0v_1\cdots$ and integers k, l we define $\pi[k..l] = v_k\cdots v_l$, $\pi[..k] = \pi[0..k]$, and $\pi[l..] = v_lv_{l+1}\cdots$. We extend the weight function to partial plays by setting $w(\pi[k..l]) = \sum_{i=k}^{l-1} w((v_i, v_{i+1}))$. A strategy for Eve (Adam) is a function σ that maps partial plays ending with a vertex v in $V_\exists$ $(V \setminus V_\exists)$ to a successor of v. A strategy has memory M if it can be realized as the output of a finite state machine with M states. A memoryless (or positional) strategy is a strategy with memory 1, that is, a function that only depends on the last element of the given partial play. A play $\pi = v_0v_1\cdots$ is consistent with a strategy σ for Eve (Adam) if whenever $v_i \in V_\exists$ $(v_i \in V \setminus V_\exists)$, $\sigma(\pi[..i]) = v_{i+1}$.

2.1 Payoff functions

A play in a game graph defines an infinite sequence of weights. We define below several common functions that map such sequences to real numbers.

Liminf/limsup. The liminf (limsup) payoff is determined by the minimum (maximum) weight seen infinitely often. Given a play $\pi = v_0v_1\cdots$ we define:

$$\liminf(\pi) = \liminf_{i\to\infty} w(v_i, v_{i+1}) \qquad \limsup(\pi) = \limsup_{i\to\infty} w(v_i, v_{i+1}).$$

Note that by negating all weights and the endpoints of the intervals we transform a limsup game to a liminf game and vice-versa.

Mean-payoff. The *mean-payoff* value of a play is the limiting average weight, however there are several suitable definitions because the running averages might not converge. The mean-payoff values of a play π we are interested in are defined as:

$$\underline{MP}(\pi) = \liminf_{k\to\infty} \frac{1}{k} w(\pi[..k]) \qquad \overline{MP}(\pi) = \limsup_{k\to\infty} \frac{1}{k} w(\pi[..k]).$$

As with liminf/limsup games we can switch between definitions by negating weights and interval endpoints, so we will only consider the $\underline{MP}$ function.

Discounted sum. The *discounted sum* is defined by a discount factor $\lambda \in (0,1)$. Given a play $\pi = v_0 v_1 \cdots$, we define:

$$DS_\lambda(\pi) = \sum_{i=0}^{\infty} \lambda^i \cdot w(v_i, v_{i+1}).$$

Total sum. The *total sum* condition can be thought of as a refinement of the mean-payoff condition, enabling discrimination between plays that have a mean-payoff of 0. Given a play π we define:

$$\underline{Total}(\pi) = \liminf_{k\to\infty} w(\pi[..k]) \qquad \overline{Total}(\pi) = \limsup_{k\to\infty} w(\pi[..k]).$$

As with liminf/limsup games we can switch between definitions by negating weights and interval endpoints, so we will only consider the $\underline{Total}$ function.

2.2 Interval games

For a fixed payoff function F, an *interval F game* consists of a finite game graph and a finite union of real intervals $I = I_1 \cup \cdots \cup I_r$ (given as a list of finitely presentable end-points). Given an interval F game (G, I), a play π in G is winning for Eve if $F(\pi) \in I$ and winning for Adam if $F(\pi) \notin I$. We say a player wins the interval game if he or she has a strategy σ such that all plays consistent with σ are winning for that player. For convenience we will assume the intervals are non-overlapping and ordered such that $\sup I_i \leq \inf I_{i+1}$ for all i.

2.3 Parity games

A parity game is a pair (G, Ω) where G is a game graph (with no weight function) and $\Omega : V \to \mathbb{N}$ is a function that assigns a priority to each vertex. Plays and strategies are defined as with interval games. A play defines an infinite sequence of priorities, and we say it is winning for Eve if and only if the minimal priority seen infinitely often is even.

3 Liminf games

The first payoff function we consider is the $\liminf$ function. Note that as this always takes integer values, we can assume all intervals are closed or open as necessary. We show below that deciding interval liminf games is polynomially equivalent to deciding parity games. In particular the number of intervals is equal to the number of even priorities required, so single interval liminf games are equivalent to parity games with at most three priorities and can therefore be solved in polynomial time [15]. Further, the range of the priorities is determined by range of the weight function and vice versa, so this equivalence also holds for unary encoded interval liminf games.

▶ **Theorem 1.** *The following problems are polynomially equivalent:*
(i) *Deciding if Eve wins a unary encoded interval liminf game;*
(ii) *Deciding if Eve wins a binary encoded interval liminf game; and*
(iii) *Deciding if Eve wins a parity game.*

Proof. (i)⇒(ii): Trivial.

(ii)⇒(iii): For this reduction, we use the following function which will also be used in Section 6. Let $I = I_1 \cup I_2 \cup \cdots \cup I_r$ be a finite union of closed integer intervals such that $\sup I_i < \inf I_{i+1}$ for all i. Define $\Omega_I : \mathbb{Z} \to [1, 2r+1]$ as follows:

$$\Omega_I(n) = \begin{cases} 2i & \text{if } n \in I_i, \\ 1 & \text{if } n < \inf I_1, \text{ and} \\ \max\{1+2i : \sup I_i < n\} & \text{otherwise.} \end{cases}$$

Now suppose (G, I) is an interval liminf game. We transform the game graph G to G' as follows. Every edge e is sub-divided and the subdividing vertex is given priority $\Omega_I(w(e))$. The original vertices of G are all given priority $2r+1$.

It is not difficult to see that there is a 1-1 correspondence between plays in G and plays in G', and that for any play in G, $\liminf w(e) \in I_i$ for some i if and only if the minimum priority in the corresponding play in G' seen infinitely often is even.

(iii)⇒(i): To go the other direction, given a parity game played on G we transform it to an interval liminf game played on G' as follows. G' is the weighted graph obtained by setting the weight of an edge to be the priority at the vertex at the tail of the edge (that is, the vertex for which the edge is outgoing). The intervals are singleton intervals containing each of the even priorities that occur in G. Clearly any play in G is a play in G' and it is not difficult to see that for a play in G the minimum priority seen infinitely often is even if and only if the lim inf of the weights of all edges in a play of G' lie in a given interval. ◀

It follows from our reduction and the positional determinacy of parity games [17], that:

▶ **Corollary 2.** *Positional strategies suffice for interval liminf games.*

4 Mean-payoff games

In this section we investigate interval mean-payoff games. We give a recursive algorithm that repeatedly asks for a solution for the *mean-payoff threshold problem.* As mentioned earlier this problem is known to be in NP ∩ coNP, and solvable in time $O(|V| \cdot |E| \cdot W)$ and space $O(|V| \cdot \log(|E| \cdot W))$ [4]. We denote this problem by $\mathbf{MP}_{\sim\nu}(G)$ where $\sim \in \{\geq, >, \leq, <\}$ depending on whether Eve is maximizing or minimizing[1] the payoff and whether or not a payoff of ν is winning for Eve. It is well known [8] that the strict threshold problem can be reduced to a non-strict threshold problem – this follows from the fact that mean-payoff values are restricted to a finite set of rationals.

Our algorithm implies that for a fixed number of intervals the problem reduces to the classic threshold problem (under polynomial-time Turing reductions). In the sequel we consider the memory requirements of single interval mean-payoff games in more detail. In particular we show that in this case finite memory strategies (indeed, positional strategies

[1] Note that by positional determinacy of mean-payoff games [8], $\underline{MP}$ and $\overline{MP}$ compute the same values when one player is maximizing the payoff. Hence our decision to use $\underline{MP}$ is not affected by whether Eve is maximizing or minimizing.

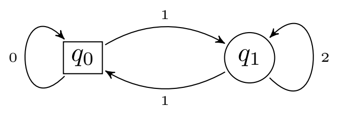


Figure 1 Interval mean-payoff game ($I = \{1\} \cup \{2\}$) which requires infinite memory.

for Adam) suffice for winning strategies. However, our first observation of this section is that in general, interval mean-payoff games may require infinite memory.

▶ **Lemma 3.** *Finite memory winning strategies are insufficient in interval mean-payoff games.*

Proof. Consider the game in Figure 1 where $I = \{1\} \cup \{2\}$. Eve has an infinite memory winning strategy in this game as follows. First she plays to q_1. Then she counts how many times Adam takes the loop (q_1, q_1). If Adam returns to q_0 then Eve takes the loop (q_0, q_0) the same number of times before returning to q_1. Clearly any play consistent with this strategy will either have $\underline{MP} = 2$ (if it eventually remains in q_1), or $\underline{MP} = 1$ (otherwise). Therefore the strategy is winning for Eve. Now suppose Eve plays a finite memory strategy σ with memory M. We observe that under this strategy Eve cannot stay at q_0 for more than M turns without staying there indefinitely. Adam's strategy is thus to stay at q_1 for $2M$ turns before returning to q_0. It is not difficult to see that any play consistent with this strategy and σ will yield a mean-payoff of either 0 or in the range $(1, 2)$, and so the strategy is winning for Adam. ◀

Upper bounds

We present an algorithm for computing the winning regions in an interval mean-payoff game in Algorithm 1. The correctness of the algorithm is given by Lemma 4.

Algorithm 1 $\mathbf{MP}_I(G)$

Input: A game graph $G = (V, V_\exists, E, w, q_0)$ and a finite union of real intervals I.
Output: $(X^\exists, X^\forall)$ where $X^\exists$ $(X^\forall)$ are the vertices from which Eve (Adam) has a winning strategy.

```
if $I = \emptyset$ then
  return $(\emptyset, V)$
end if
$a \leftarrow \inf I$
if $a = -\infty$ then
  $(X, X') \leftarrow \mathbf{MP}_{\mathbb{R}\setminus I}(\overline{G})$          {$\overline{G}$ is $G$ with $V_\exists$ and $V \setminus V_\exists$ swapped}
else
  $X \leftarrow \emptyset$
  repeat
    $(A, A') \leftarrow \mathbf{MP}_{\succ a}(G)$          {If $a \in I$ then $\succ = \geq$, otherwise $\succ = >$}
    $(B, B') \leftarrow \mathbf{MP}_{(-\infty,a]\cup I}(G)$
    $X \leftarrow X \cup A' \cup B'$
    $G \leftarrow G \setminus (A' \cup B')$
  until $A' \cup B' = \emptyset$
end if
return $(V \setminus X, X)$
```

▶ **Lemma 4.** *Let (G, I) be an interval mean-payoff game.* $\boldsymbol{MP}_I(G)$ *correctly computes the winning regions for Adam and Eve.*

Proof sketch. We observe that at each iteration of the loop we remove vertices from which Adam can either ensure the mean-payoff lies below I or from which he can (recursively) win on the larger set (with fewer interval end-points) $I' = (-\infty, \inf I] \cup I$. Thus it is clear that Adam has a winning strategy from any vertex removed. We now argue that Eve has a winning strategy on any vertex remaining. Observe that from these vertices she has two strategies: a positional strategy $\sigma_>$ which ensures $\underline{MP} \succ \inf I$ and $\sigma_<$ which wins with the larger objective I'. We combine these into a winning strategy for I as follows. Eve chooses any t in the first interval and keeps track of the current average weight. Whenever the average weight lies below t she plays $\sigma_>$, and whenever it lies above t she plays $\sigma_<$. As the mean-payoff objective is a tail objective and therefore independent of an initial play prefix, it is not difficult to see that if she eventually plays only one strategy then the mean-payoff will lie in I. If she changes strategy infinitely often, then the fact that $\sigma_>$ is positional means she will never deviate too far below $\inf I$ so the $\liminf$ average will lie between $\inf I$ and t, and thus in I. ◀

The running time for Algorithm 1 is $|V|^{2r-1} \cdot \mathbf{MP}$, where $\mathbf{MP}$ is the running time for an algorithm to solve the mean-payoff threshold problem. It is straightforward to see that the algorithm can be implemented in polynomial space.

▶ **Theorem 5.** *Let G be a game graph and I a finite union of r real intervals. Whether Eve wins the interval mean-payoff game (G, I) can be decided in time $O(|V|^{2r} \cdot |E| \cdot W)$ and space $O(r \cdot |V| \cdot \log(|E| \cdot W))$.*

▶ **Corollary 6.** *Determining if Eve wins an interval mean-payoff game for a fixed number of intervals is in* NP ∩ coNP*.*

We observe that although the players may require infinite memory for a winning strategy, Algorithm 1 shows that a winning strategy can be succinctly represented by $2r$ positional sub-strategies. It is not clear that given such a certificate whether there exists an efficient algorithm for computing the winning region, however we believe that this is the case. By the symmetry of the roles of the players, such an algorithm would show that the interval mean-payoff game is both in NP and coNP.

▶ **Conjecture 1.** *Determining if Eve wins an interval mean-payoff game is in* NP ∩ coNP*.*

Lower bound

The above conjecture would hold if we could solve interval mean-payoff games with only polynomially many calls to the mean-payoff threshold problem. We now give a lower bound for the complexity of deciding interval mean-payoff games which suggests any such algorithm would yield quite remarkable results: we reduce parity games to interval mean-payoff games with small weights and interval bounds. In particular this implies that any pseudo-polynomial time algorithm (including polynomially many calls to the threshold problem) would yield a polynomial time algorithm for parity games.

▶ **Theorem 7.** *There is a polynomial time reduction from parity games to unary encoded interval mean-payoff games.*

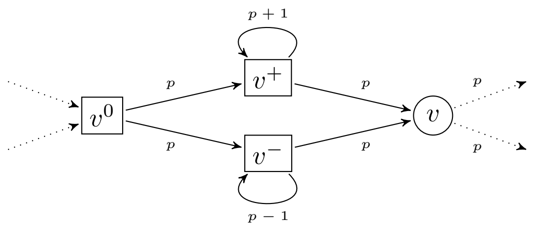


Figure 2 Vertex gadget for vertex $v \in V \setminus V_\exists$ with even priority p.

Proof sketch. Intuitively, we replace each vertex in the original game with the gadget shown in Figure 2. If the priority of the vertex is even then the gadget is controlled by Eve, and if it is odd then it is controlled by Adam. The last vertex in the gadget is controlled by the player that controlled the original vertex. Given a winning strategy in the parity game, Eve's corresponding strategy is to play the same on the original vertices, and whenever the play reaches a gadget corresponding to an even priority p she remains in the gadget until the current average lies in the interval $[p, p + \frac{1}{2}]$. This ensures that the minimum average seen infinitely often corresponds to the minimum priority seen infinitely often and so, with the interval set $I = \bigcup_{p \text{ even}} [p, p + 1)$, the strategy will be winning for Eve. Conversely, Adam can translate winning strategies from the parity game in the same manner. ◀

Memory requirements for single interval mean-payoff games

The strategies for Adam and Eve described in the proof of Lemma 4 require infinite memory. We now show, with a careful analysis, that in the case of a single interval this can be improved. Note that as we can replace any strict threshold call with a non-strict threshold we can assume without loss of generality that I is closed.

▶ **Theorem 8.** *Let (G, I) be a single interval mean-payoff game. If Adam has a winning strategy then he has a positional winning strategy. If Eve has a winning strategy then she has a strategy that requires finite memory.*

Proof sketch. The fact that Adam only requires positional strategies follows from [13] and the observation that in the single interval case the winning condition is convex.

The idea behind Eve's finite memory strategy on $X^\exists$ is to keep track of the total weight seen so far (rather than the average as in Lemma 4) *modulo cycles with average weight in I*. The intuition is that such cycles do not affect whether the overall mean-payoff will lie in the interval, so we can safely ignore them. As $\sigma_<$ and $\sigma_>$ are both positional in this case, by ignoring these cycles the total weight will never deviate too far from a central value, so finite memory suffices. ◀

5 Discount sum games

In this section we consider interval discount sum games. Here we make a distinction between whether or not singleton intervals (and singleton gaps between intervals) are permitted, because unlike other payoff functions considered in this paper there is a marked difference between the corresponding games. We show that for non-singleton intervals the problem of determining the winner is PSPACE-complete and as a consequence of our algorithm we show that finite memory strategies suffice. For singleton intervals (including the exact value problem) our PSPACE-hardness result holds, but is not even known if determining the winner is decidable.

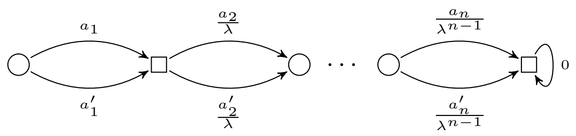


Figure 3 Reduction from subset sum games to interval discount sum games.

5.1 Single, non-singleton intervals

Lower bound

To show PSPACE-hardness we reduce from the subset sum game defined in [9]. A subset sum game is specified by a target $t \in \mathbb{N}$ and a list of pairs of natural numbers $(a_1, a'_1), \ldots, (a_n, a'_n)$. The game takes n rounds, in round i, one player (Adam if i is odd, Eve if i is even) chooses a_i or a'_i. After n rounds Eve wins if and only if the sum of the selected numbers is t. Given an instance of the subset sum game we construct the interval discount sum game shown in Figure 3, with interval $[t, t+1)$.

It is clear that a play in this game corresponds to a selection of elements from the pairs, and the discounted sum of the play is equal to the sum of the corresponding elements. As this sum is always an integer, the discounted sum lies in the interval $[t, t+1)$ if and only if the selected sum is equal to t.

Upper bound

Given $v \in V$ and strategies σ and τ for Eve and Adam respectively, we define $\mathrm{v}^v_{\sigma\tau}$ to be the payoff of the unique play from v consistent with σ and τ. Two important (memoryless) strategies for Eve are $\sigma_{\max}$ and $\sigma_{\min}$, the strategies which, for all states v, maximize $\min_\tau \mathrm{v}^v_{\sigma\tau}$ and minimize $\max_\tau \mathrm{v}^v_{\sigma\tau}$ respectively.

The idea behind the upper bound centres around the observation that after sufficiently many steps the remainder of any play does not contribute much to the overall discounted sum. If the target interval is non-singleton then the problem reduces to the classical threshold problem. Thus we can stop the game after finitely many steps when it becomes a trivial matter to determine if the overall discounted sum will lie in the interval or not. The key lemma for the result, the proof of which we defer to the full paper, is the following:

▶ **Lemma 9.** *Let (G, I, λ) be an interval discount sum game that Eve wins, and let*

$$N = \left\lfloor \frac{\log(\sup I - \inf I) + \log(1-\lambda) - \log(2W)}{\log \lambda} \right\rfloor .$$

Then Eve has a winning strategy that agrees with either $\sigma_{\max}$ or $\sigma_{\min}$ after N steps.

Note that whether the strategy agrees with $\sigma_{\max}$ or $\sigma_{\min}$ depends on the play up to the N-th step. It is feasible that against one strategy of Adam this strategy will agree with $\sigma_{\max}$ but against another strategy it will agree with $\sigma_{\min}$.

▶ **Corollary 10.** *Finite memory strategies suffice in non-singleton interval discount sum games.*

The algorithm for determining the winner of a non-singleton interval discount sum game is straightforward. We run an alternating Turing Machine for N steps to guess an initial play.

Note that N is polynomial in the size of the input, so this can be done in PSPACE. Suppose the play ends in state v with the current discounted sum S. We compute the four values:

$$\overline{M} = \max_{\tau} \mathrm{v}^{v}_{\sigma_{\max}\tau} \quad \underline{M} = \min_{\tau} \mathrm{v}^{v}_{\sigma_{\max}\tau} \quad \overline{m} = \max_{\tau} \mathrm{v}^{v}_{\sigma_{\min}\tau} \quad \underline{m} = \min_{\tau} \mathrm{v}^{v}_{\sigma_{\min}\tau}.$$

These are computable in $\mathsf{NP} \cap \mathsf{coNP}$ using an algorithm that computes values in classic discount sum games [18]. Finally we check if either:

$$\begin{array}{lll} S + \lambda^{N+1} \cdot \underline{M} \in I & \text{and} & S + \lambda^{N+1} \cdot \overline{M} \in I, \text{ or} \\ S + \lambda^{N+1} \cdot \underline{m} \in I & \text{and} & S + \lambda^{N+1} \cdot \overline{m} \in I. \end{array}$$

It is clear that one of the above conditions holds if and only if $\sigma_{\max}$ or $\sigma_{\min}$ is winning from the current position. Therefore, from Lemma 9, one of the above conditions holds if and only if Eve has a winning strategy.

▶ **Theorem 11.** *Let G be a game graph, $I \subseteq \mathbb{R}$ a non-singleton interval, and $\lambda \in (0, 1)$. Deciding if Eve wins the interval discount sum game (G, I, λ) is* PSPACE*-complete.*

We observe that if the weights, interval bounds and discount factor are all encoded in unary then N is logarithmic in the size of the input and $\overline{M}$, $\underline{M}$, $\overline{m}$, and $\underline{m}$ can all be computed in polynomial time using a pseudo-polynomial time algorithm for discount sum games [18]. Thus the above algorithm runs in polynomial time.

▶ **Theorem 12.** *Let G be a game graph, $I \subseteq \mathbb{R}$ a non-singleton interval and $\lambda \in (0, 1)$. Deciding if Eve wins the unary encoded interval discount sum game (G, I, λ) is in* P.

5.2 Multiple intervals

The algorithm of the previous section also applies to multiple intervals *as long as the gaps between the intervals are also non-singleton.* This follows from the observation that after sufficiently many steps the overall discount payoff will not deviate too far from the current value, so at that point the game reduces to the single interval case.

▶ **Theorem 13.** *Let G be a game graph, I a finite union of real intervals such that neither I nor $\mathbb{R} \setminus I$ contains singleton elements, and $\lambda \in (0, 1)$. Deciding if Eve wins the interval discount sum game (G, I, λ) is* PSPACE*-complete.*

5.3 Singleton intervals

When the set of intervals (or their complement) include singleton intervals, the situation is more complicated. Following the same argument as the previous section, after sufficiently many steps the problem reduces to the exact value problem: Given a discount sum game (G, λ) and a target $t \in \mathbb{Q}$, does Eve have a strategy to ensure the discounted sum is exactly t?

It is currently open whether this problem is even decidable, however the PSPACE-hardness result from the previous section (using the interval $\{t\}$ rather than $[t, t + 1)$) gives a lower-bound. The problem is related to the universality problem for discount sum automata [2], a well-known problem for which decidability remains open [1]. The problem was also studied for Markov Decision Processes and graphs (i.e. one-player games) in [7] where it was shown to be decidable for discount factors of the form $\lambda = \frac{1}{n}$, and that in general infinite memory is required for winning strategies.

▶ **Lemma 14** ([7])**.** *There exist exact value discount sum games for which an infinite memory is required for a winning strategy.*

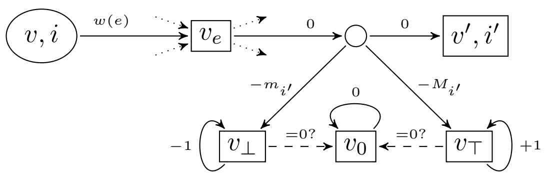


Figure 4 Edge gadget for edge $e = (v, v')$, $v \notin V_\exists$, $v' \in V_\exists$.

6 Total sum games

Total sum games refine mean-payoff games and can be seen as a special case of discount sum games where the discount factor is 1. Assuming the graph has integer weights, $\underline{Total}$ will always be an integer (or $\pm\infty$), thus we can assume all intervals are closed or open as necessary.

In this section we show that determining the winner of such games is PSPACE-complete for unary encoded games and EXPSPACE-complete for binary encoded games. Our bounds are obtained by relating interval total sum games with various one-counter games, that is, games played on the transition graph of a one-counter machine (equivalently, one-dimensional vector addition systems with states). Intuitively a one-counter game graph is a game graph augmented with a counter which is incremented or decremented by weights on traversed edges. A special set of edges are activated only if the counter has value 0. We give a reduction from one-counter reachability games, studied in [3, 14, 12] to establish lower bounds, and a reduction to one-counter parity games, studied in [16], for the upper bounds.

Lower bounds

We give a reduction from counter reachability games to (singleton) interval total sum games. This establishes the necessary lower bounds because determining the winner in such games was shown to be PSPACE-complete for unary encoded games in [14] and EXPSPACE-complete for binary encoded games in [12], even for the restricted one-counter game graphs considered here.

Given a one-counter reachability game with no zero-activated edges and target set $F \subseteq V_\exists$, we construct a game graph as follows. We double the weights of all edges; add a new initial vertex with an edge of weight $+1$ to the original initial vertex; and add a new global sink with a self loop of weight 0 and edges of weight -1 from all vertices in F. By parity arguments the new vertices are the only vertices where the total sum can be 0, and Eve has a strategy to reach the global sink if and only if she can reach F in the original game with counter value 0. Thus Eve wins the interval total sum game with interval $\{0\}$ if and only if she wins the original one-counter reachability game.

Upper bounds

We now show that interval total sum games can be solved in EXPSPACE by reducing them to parity games on infinite graphs defined by the transition graphs of one-counter machines. Such games were considered in [16], where a PSPACE algorithm was given for unary weighted one-counter games (equivalently, single alphabet pushdown processes). As a binary one-counter graph can be described by an exponentially larger unary one-counter graph, our reduction yields an EXPSPACE algorithm.

The key observation for the reduction is that interval total sum games can be viewed as parity games on $V \times \mathbb{Z}$, where the second component indicates the total sum so far. The

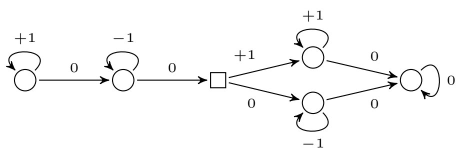


Figure 5 Interval total sum game ($I = \mathbb{R} \setminus \{0\}$) which requires infinite memory.

priority of a vertex (v, c) is determined by which interval (or gap between intervals) contains c, in the same manner used in the equivalence between liminf games and parity games in Section 3. However, we cannot use the result of [16] directly because for those games the priorities are defined by the states of the counter-machine and not the values of the counter. Instead, we have Eve assert which interval (or gap between intervals) the counter is in, and give Adam the ability to punish her if she claims falsely.

Let (G, I) be an interval total sum game. Recall from Section 3 the definition of Ω_I. Let us define $m_i := \min \Omega_I^{-1}(i)$ and $M_i := \max \Omega_I^{-1}(i)$. Intuitively, the reduction creates $2r + 1$ copies of the G (one for each interval and one for each gap), but we replace edges with the edge gadget shown in Figure 4. The priority of a vertex (v, i) is i, and for each gadget vertex it is $2r + 1$ except for v_0 which has priority $2r$.

We now show that Eve has a winning strategy in this parity game if and only if she has a winning strategy in the interval total sum game. We first observe that if the play reaches the predecessor of (v, i) and the counter value is outside $[m_i, M_i]$ then Adam can win by playing to $v_\perp$ if the counter is $< m_i$ or to $v_\top$ if the counter is $> M_i$. On the other hand, if the counter is in the range $[m_i, M_i]$ then Eve wins if Adam plays to either of these vertices. Thus the gadget defined by the vertices $\{v_\perp, v_\top, v_0\}$ allows Adam to punish Eve if and only if the counter is not in the asserted interval. Now, assuming Eve plays correctly, it is easy to see that the minimal priority seen infinitely often corresponds to the lowest interval or interval gap visited infinitely often by the counter. Thus Eve has a winning strategy in the parity game if and only if she has a winning strategy in the interval game.

▶ **Theorem 15.** *Deciding if Eve wins a binary (unary) encoded interval total sum game is* EXPSPACE*-complete (*PSPACE*-complete).*

Memory requirements

We now consider memory requirements for winning strategies in interval total sum games. Figure 5 shows that, in general, infinite memory is required for winning strategies.

▶ **Lemma 16.** *Finite memory winning strategies are insufficient in interval total sum games.*

By swapping the roles of the players and complementing the interval, we see that even for single interval games Adam may require infinite memory. We now show this is not the case for Eve. Indeed, using König's lemma, we can show a more general result.

▶ **Lemma 17.** *Let (G, I) be an interval total sum game where $I \cap \mathbb{Z}$ is finite. If Eve has a winning strategy then she has a finite memory winning strategy.*

To complete the argument for single interval total sum games, we observe that if the interval is infinite then we are considering the classical threshold problem for total sum games. Positional strategies for these games were shown to be sufficient in [11].

▶ **Theorem 18.** *Let (G, I) be a single interval total sum game. If Eve has a winning strategy then she has a finite memory winning strategy.*

References

1 U. Boker and T. A. Henzinger. Determinizing discounted-sum automata. In *CSL*, pages 82–96, 2011.

2 U. Boker and J. Otop. Personal communcation, 2014.

3 T. Brázdil, P. Jancar, and A. Kucera. Reachability games on extended vector addition systems with states. In *ICALP (2)*, pages 478–489, 2010.

4 L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.

5 K. Chatterjee, L. Doyen, and T. A. Henzinger. A survey of partial-observation stochastic parity games. *Formal Methods in System Design*, 43(2):268–284, 2013.

6 K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In *Proc. of FSTTCS*, pages 505–516, 2010.

7 K. Chatterjee, V. Forejt, and D. Wojtczak. Multi-objective discounted reward verification in graphs and mdps. In *LPAR*, pages 228–242, 2013.

8 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

9 J. Fearnley and M. Jurdzinski. Reachability in two-clock timed automata is pspace-complete. In *ICALP*, volume 2, pages 212–223, 2013.

10 T. Gawlitza and H. Seidl. Games through nested fixpoints. In *CAV*, pages 291–305, 2009.

11 H. Gimbert and W. Zielonka. When can you play positionally? In *MFCS*, pages 686–697, 2004.

12 P. Hunter. Reachability in succinct one-counter games. Available at `http://arxiv.org/abs/1407.1996`, 2014.

13 E. Kopczynski. Omega-regular half-positional winning conditions. In *CSL*, pages 41–53, 2007.

14 J. Reichert. On the complexity of counter reachability games. In *RP*, pages 196–208, 2013.

15 S. Schewe. Solving parity games in big steps. In *FSTTCS*, pages 449–460, 2007.

16 O. Serre. Parity games played on transition graphs of one-counter processes. In *FoSSaCS*, pages 337–351, 2006.

17 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.

18 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1):343–359, 1996.

Playing Safe*

Thomas Colcombet[1], Nathanaël Fijalkow[1,2], and Florian Horn[1]

1 **LIAFA, Université Paris 7 – Denis Diderot, France**
2 **University of Warsaw, Poland**

Abstract

We consider two-player games over graphs and give tight bounds on the memory size of strategies ensuring safety conditions. More specifically, we show that the minimal number of memory states of a strategy ensuring a safety condition is given by the size of the maximal antichain of left quotients with respect to language inclusion. This result holds for all safety conditions without any regularity assumptions, and for all (finite or infinite) graphs of finite degree.

We give several applications of this general principle. In particular, we characterize the exact memory requirements for the opponent in generalized reachability games, and we prove the existence of positional strategies in games with counters.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Game Theory, Synthesis, Safety Specifications, Program Verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.379

1 Introduction

Graphs games provide a mathematical framework to automatically address many questions, for instance they are used to model reactive systems (we refer to [9] for a survey on the topic). We focus here on the Synthesis Problem to motivate the problem we consider, which is to characterize the amount of memory required in games with safety conditions.

The Synthesis Problem. The inputs of the Synthesis Problem are a *system* and a *specification*. The expected output is a *controller* for the system, that ensures the specification.

We describe here an approach to solve the Synthesis Problem through Game Theory. We model the system as a graph, whose vertices represent states and edges represent transitions. Its evolution consists in interactions between a controller and an environment, which is turned into a game on the graph between two players, Eve and Adam. If in a given state, the controller can choose the evolution of the system, then the corresponding vertex is controlled by Eve. If the system evolves in an uncertain way, we consider the worst-case scenario, where Adam controls those states.

A pebble is initially placed on the vertex representing the initial state of the system, then Eve and Adam move this pebble along the edges. The sequence built describes a run of the system: Eve tries to ensure that it satisfies the specification.

So, in order to synthesize a controller, we are interested in whether Eve can ensure this objective and what resources she needs. In particular, the most salient question is: what is the size of a *minimal* controller satisfying the specification? Since a controller is here

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454 (GALE) and n° 239850 (SOSNA).

given by a strategy for Eve, this is equivalent to the following question: what is the minimal amount of memory used by a winning strategy?

The following diagram shows the correspondence between the notions from the Synthesis Problem (left-hand side) and the game-theoretical notions (right-hand side).

$$\underbrace{\mathbb{S}}_{\text{system}}, \underbrace{\mathcal{C}}_{\text{controller}} \models \underbrace{\Phi}_{\text{specification}} \qquad \Longleftrightarrow \qquad \underbrace{\mathcal{A}}_{\text{arena (graph)}}, \underbrace{\sigma}_{\text{strategy}} \models \underbrace{W}_{\text{condition}}$$

Safety Specifications. Since we consider non-terminating sequences, a specification is given by a language of infinite words. Of special interest are the specifications asserting that "nothing bad" ever happens; such specifications are called *safety specifications*. A safety specification is induced by a (possibly infinite and non-regular) set of bad prefixes P, and the specification is met by a run if no prefixes belong to P.

Although quite simple, the safety specifications proved useful in both theory and practice, and are actively studied (we refer to [12] for a survey).

Our contribution. In this paper, we show the following general principle:

For a safety condition W, the minimal number of memory states of a winning strategy is exactly the cardinal of the maximal antichain of left quotients of W.

We refer to Section 3 for the missing definitions. Note that this result holds for all safety conditions, without any regularity assumption. The only assumption made is that graphs have finite degree: we prove that this is necessary, by providing a counter example not satisfying this property. The Section 3 is devoted to the proof of this main result. We give several examples and applications in Section 4. For instance, it allows to characterize the memory requirements for the opponent in generalized reachability games, and to prove the existence of positional strategies in games with counters.

Related Works: Evaluating Memory Requirements. Characterizing the amount of memory required by winning strategies according to the winning conditions has been widely studied, in different frameworks.

The first result in this direction is about Muller conditions: in [5], the authors show how to compute the exact memory requirements by looking at the so-called Zielonka tree of a Muller condition. This is orthogonal to our results, as the Muller conditions only specify the limit behaviour (what is seen infinitely often), whereas we consider here only the behaviours in the finite.

In a different direction, Hugo Gimbert and Eryk Kopczyński independently investigated necessary and sufficient conditions for positional determinacy. We refer to their respective PhD theses [8, 11] for more details. A submitted result of Eryk Kopczyński [10] shows that one can compute the exact *chromatic* memory requirements of ω-regular conditions. Our results are also orthogonal to this result, as we characterize the exact memory requirements for all safety conditions, including *non-regular* ones.

2 Definitions

The games we consider are played on an *arena* $\mathcal{A} = (V, (V_\exists, V_\forall), E, c)$, consisting of a (finite or infinite) graph (V, E), a partition $(V_\exists, V_\forall)$ of the vertex set V: a vertex in $V_\exists$ belongs to

Eve and in $V_\forall$ to Adam, and a coloring function $c : E \to A$ mapping edges to a color from a finite alphabet A. When drawing arenas, we will use circles for vertices owned by Eve and squares for those owned by Adam. Throughout this paper, we make two assumptions:

- There are no dead-ends: for every vertex $v \in V$, there exists an edge $(v, v') \in E$;
- The degree is finite: for every vertex $v \in V$, the set $\{v' \mid (v, v') \in E\}$ is finite.

The first assumption is cosmetic. The second assumption, however, will be crucial: in particular, we will give a counter-example showing that our results do not hold without this assumption.

Game. A *play* π is an infinite word of edges $e_0 \cdot e_1 \cdots$ that are consecutive: for all i, $e_i = (\_, v) \in E$ and $e_{i+1} = (v, \_) \in E$ for some $v \in V$. We denote π_k the prefix of length k of π. A play π induces an infinite sequence of colors $c(\pi)$, obtaining by applying the coloring function c component-wise. We define *winning conditions* for a player by giving a set of infinite sequences of colors $W \subseteq A^\omega$. As we are interested in zero-sum games, *i.e.* where the winning conditions of the two players are opposite, if the winning condition for Eve is W, then the winning condition for Adam is $A^\omega \setminus W$. A *game* is a couple $\mathcal{G} = (\mathcal{A}, W)$ where $\mathcal{A}$ is an arena and W a winning condition.

Strategy. A *strategy* for a player is a function that prescribes, given a finite history of the play, the next move. Formally, a strategy for Eve is a function $\sigma : E^* \cdot V_\exists \to E$ such that for all $\pi \in E^*$ and $v \in V_\exists$ we have $\sigma(\pi \cdot v) = (v, \_) \in E$. Strategies for Adam are defined similarly, and usually denoted τ. Once a game $\mathcal{G} = (\mathcal{A}, W)$, a starting vertex v_0 and strategies σ for Eve and τ for Adam are fixed, there is a unique play $\pi(v_0, \sigma, \tau)$, which is said winning for Eve if its image by c belongs to W. A strategy σ for Eve is *winning* if for all strategies τ for Adam, $\pi(v_0, \sigma, \tau)$ is winning. We say that Eve wins the game $\mathcal{G}$ from v_0 if she has a winning strategy from v_0, and denote $\mathcal{W}_E(\mathcal{G})$ the set of vertices from where Eve wins; we often say that $v \in \mathcal{W}_E(\mathcal{G})$ is winning. We define similarly $\mathcal{W}_A(\mathcal{G})$ for Adam to be the set of vertices from where Adam wins.

Memory. A *memory structure* is a deterministic state machine that reads the sequence of edges and abstracts its relevant informations into a memory state. Formally, a memory structure $\mathcal{M} = (M, m_0, \mu)$ for an arena consists of a set M of memory states, an initial memory state $m_0 \in M$ and an update function $\mu : M \times E \to M$. The update function takes as input the current memory state and the chosen edge to compute the next memory state. It can be extended to a function $\mu^* : E^* \to M$ by defining $\mu^*(\varepsilon) = m_0$ and $\mu^*(\pi \cdot e) = \mu(\mu^*(\pi), e)$. Given a memory structure $\mathcal{M}$ and a next-move function $\nu : V_\exists \times M \to E$, we can define a strategy σ for Eve by $\sigma(\pi \cdot v) = \nu(v, \mu^*(\pi \cdot v))$. A strategy with memory structure $\mathcal{M}$ has finite memory if M is a finite set. It is *memoryless*, or *positional* if M is a singleton: it only depends on the current vertex. Hence a memoryless strategy can be described as a function $\sigma : V_\exists \to E$. We denote $\mathrm{mem}(\sigma)$ the number of memory states used by the strategy σ.

An arena and a memory structure induce an expanded arena where the current memory state is computed online. Formally, the arena $\mathcal{A} = (V, (V_\exists, V_\forall), E, c)$, the memory structure $\mathcal{M}$ for $\mathcal{A}$ and a new coloring function $c' : E \times M \to A$ induce an expanded arena $\mathcal{A} \times \mathcal{M} = (V \times M, (V_\exists \times M, V_\forall \times M), E \times \mu, c')$, where $E \times \mu$ is defined by: $((v, m), (v', m')) \in E \times \mu$ if $(v, v') \in E$ and $\mu(m, (v, v')) = m'$. From a memoryless strategy in $\mathcal{A} \times \mathcal{M}$, we can build a strategy in $\mathcal{A}$ using $\mathcal{M}$ as memory structure, which behaves as the original strategy. This key observation will be used several times in the paper.

3 Tight Bounds on the Memory for Safety Conditions

In this section, we consider a safety condition[1] W and compute the following quantity:

$$\mathrm{mem}(W) \doteq \sup_{\mathcal{G}=(\mathcal{A},W)\text{ game}} \inf_{\substack{\sigma \text{ winning} \\ \text{strategy}}} \mathrm{mem}(\sigma) \ .$$

In words, $\mathrm{mem}(W)$ is the necessary and sufficient number of memory states for constructing a winning strategy in games with condition W. Equivalently:

- *upper bound:* for all games $\mathcal{G} = (\mathcal{A}, W)$, if Eve wins, then she has a winning strategy using at most $\mathrm{mem}(W)$ memory states,
- *lower bound:* there exists a game $\mathcal{G} = (\mathcal{A}, W)$ when Eve wins but she has no winning strategy using less than $\mathrm{mem}(W)$ memory states.

The reader with a background in game theory may be surprised, as it is well known that "safety games are positionally determined", implying that the quantity above is constant equal to one. The subtlety here is that our setting is (much) more general than the classical notion of safety games. Specifically, consider an arena $\mathcal{A}$:

- an *internal* safety condition is given by a subset $B \subseteq A$ of forbidden colors, inducing the winning condition

$$\mathrm{Safe}(B) = \{a_0 \cdot a_1 \cdots \in A^\omega \mid \text{for all } i,\ a_i \notin B\} \ ,$$

- an *external* safety condition is given by a subset $P \subseteq A^*$ of forbidden prefixes of colors, inducing the winning condition

$$\mathrm{Safe}(P) = \{a_0 \cdot a_1 \cdots \in A^\omega \mid \text{for all } i,\ a_0 a_1 \cdots a_i \notin P\} \ .$$

The term "internal" refers to the idea that the set B can be thought of as a set of forbidden edges in the graph, giving rise to the classical notion of safety games, where Eve tries to ensure never to reach the forbidden parts in the graph.

▶ **Lemma 1** (Folklore). *Let $\mathcal{G}$ be a game with an internal safety condition. If Eve has a winning strategy, then she has a positional winning strategy.*

On the other hand, the external safety conditions describe much more, as we will demonstrate in Section 4.

From now on, as we are mostly interested in external safety conditions, we will drop the prefix "external". The notion of safety condition originates from topological studies of the set of infinite words: the safety conditions are the closed sets for the Cantor topology, denoted Π_1 in the corresponding Borel hierarchy.

For the remainder of this section, we fix a safety condition $W = \mathrm{Safe}(P)$ induced by $P \subseteq A^*$.

A First Upper Bound

We first give an upper bound on $\mathrm{mem}(W)$. Let $w \in A^*$, define its left quotient as:

$$w^{-1}W = \{\rho \in A^\omega \mid w \cdot \rho \in W\}.$$

[1] To be defined in this section.

We denote $\mathrm{Res}(W)$ the set of left quotients of W. We mention some special left quotients: the initial one, $\varepsilon^{-1}W$ (equal to W), and the empty one, obtained as $w^{-1}W$ for any $w \in P$. From a left quotient $w^{-1}W$ and a letter $a \in A$, we define $(w^{-1}W) \cdot a$ as $(w \cdot a)^{-1}W$: it is easy to check that this is well defined (independent of the representant w chosen). Recall that $\mathrm{Res}(W)$ is finite if and only if W is regular, and in such case it can be used to describe the set of states of the minimal deterministic automaton recognizing W.

▶ **Lemma 2.** *Let $\mathcal{G} = (\mathcal{A}, W)$ be a game with a safety condition W. If Eve wins, then she has a winning strategy using at most $|\mathrm{Res}(W)|$ memory states.*

Consequently, for all safety conditions W,

$$\mathrm{mem}(W) \ \leq\ |\mathrm{Res}(W)|\ .$$

Proof. We construct a memory structure $\mathcal{M}$, as follows: $\mathcal{M} = (\mathrm{Res}(W), W, \nu)$, where $\nu(w^{-1}W, a) = (w^{-1}W) \cdot a$. At any point in the game, the memory state computed by $\mathcal{M}$ is the current left quotient. We construct the expanded arena $\mathcal{A} \times \mathcal{M}$ equipped with the coloring function $c' : E \times \mathrm{Res}(W) \to \{0, 1\}$ defined by:

$$c'(\_, w^{-1}W) = \begin{cases} 0 & \text{if } w^{-1}W = \emptyset \text{ (equivalently, } w \in P)\ , \\ 1 & \text{otherwise}\ . \end{cases}$$

We attach to $\mathcal{A} \times \mathcal{M}$ the *internal* safety condition induced by $B = \{0\}$, giving rise to the game $\mathcal{G} \times \mathcal{M} = (\mathcal{A} \times \mathcal{M}, \mathrm{Safe}(B))$. First observe that by construction, a play in $\mathcal{A} \times \mathcal{M}$ is of the form

$$(e_0, c(e_0)^{-1}W) \cdot (e_1, c(e_0 \cdot e_1)^{-1}W) \cdots (e_k, c(e_0 \cdot e_1 \cdots e_k)^{-1}W) \cdots\ ,$$

so by definition of c' a play is winning is $\mathcal{G} \times \mathcal{M}$ if and only if its projection (on the first component) is winning in $\mathcal{G}$.

It follows that a winning strategy for Eve in $\mathcal{G}$ from v_0 induces a winning strategy in $\mathcal{G} \times \mathcal{M}$ from (v_0, W). Now, thanks to Lemma 1, since Eve wins in $\mathcal{G} \times \mathcal{M}$, she has a positional winning strategy. This induces a winning strategy in $\mathcal{G}$ using $\mathcal{M}$ as memory structure, concluding the proof of Lemma 2. ◀

The game $\mathcal{G} \times \mathcal{M}$ defined above will be an important tool in the proofs to follow. We will also rely on the following remark: let $\mathcal{G} = (\mathcal{A}, W)$ be a game with a safety condition W, and assume we want to prove that a strategy σ in $\mathcal{G}$ is winning. Then it is enough to show that for all plays π consistent with σ, for all k, $c(\pi_k)^{-1}W \neq \emptyset$, where π_k is the prefix of π of length k. This simple observation follows from the definition of safety conditions.

A Tighter Upper Bound

The memory structure $\mathcal{M}$ defined above is not optimal. A first remark is that the empty left quotient (which exists if $W \neq A^\omega$) can be removed from the memory states as the game is lost. From now on by "left quotient" we mean "non-empty left quotient of W", and in particular $\mathrm{Res}(W)$ denotes the set of non-empty left quotients of W.

The second remark is the following: let L_1 and L_2 be two left quotients of W, such that $L_1 \subseteq L_2$. With the same notations as above, consider a vertex v in the arena $\mathcal{A}$. If Eve wins from (v, L_1) in $\mathcal{G} \times \mathcal{M}$, then she also wins from (v, L_2): indeed, she can play as she would have played from (v, L_1). Since this ensures from v that all plays are winning for L_1, then a fortiori they are winning for L_2.

This suggests to restrict the memory states only to *minimally winning* left quotients with respect to inclusion. Two issues arise:

- which left quotients are winning depends on the current vertex, so the semantics of a memory state can no longer be *one* left quotient, but rather a left quotient for each possible vertex,
- there may not exist *minimally winning* left quotients.

For the sake of presentation, we first show how to deal with the first issue, assuming the second issue does not appear. Specifically, in the following lemma, we assume that $\mathrm{Res}(W)$ is finite (*i.e.* W is regular), implying the existence of minimally winning left quotients. We will later drop this assumption.

We define the width of an ordered set $(E, \leq)$ as the cardinal of the maximal antichain of E with respect to $\leq$, *i.e.* the cardinal of the largest set of pairwise incomparable elements.

▶ **Lemma 3** (Upper bound in the regular case). *Let $\mathcal{G} = (\mathcal{A}, W)$ be a game with a safety condition W. Assume that* $\mathrm{Res}(W)$ *is finite, i.e. that W is regular.*

If Eve wins, then she has a winning strategy using at most K memory states, where K is the width of $(\mathrm{Res}(W), \subseteq)$.

Proof. We use the same notations as for the proof of Lemma 2, and construct a smaller memory structure together with a winning strategy using this memory structure. In this proof, by winning we mean winning in the game $\mathcal{G} \times \mathcal{M}$.

Let K be the cardinal of the maximal antichain of left quotients of W. We construct the memory structure $\mathcal{M}^* = (\{1, \ldots, K\}, 1, \mu)$, and the strategy σ induced by the next-move function ν.

Let v be a vertex in $\mathcal{A}$. We consider the set of minimal left quotients L such that (v, L) is winning. (Here we use the finiteness of $\mathrm{Res}(W)$ to guarantee the existence of such left quotients.) This is an antichain, so there are at most K of them, we denote them by $L_1(v), \ldots, L_p(v)$, for some $p \leq K$. The *key* property is that for every left quotient L such that (v, L) is winning, there exists i such that $L_i(v) \subseteq L$. Furthermore, we choose $L_1(v_0)$ such that $L_1(v_0) \subseteq W$. (Indeed, by assumption (v_0, W) is winning.)

We define the update function: $\mu(i, (v, v'))$ is a j such that $L_j(v') \subseteq L_i(v) \cdot c(v, v')$. Note that in general, such a j may not exist; it does exist if $(v', L_i(v) \cdot c(v, v'))$ is winning, and we will prove that this will always be the case when playing the strategy σ.

We define the next-move function ν (inducing σ). Let $v \in V_\exists$, and consider $(v, L_i(v))$: since Eve wins from there, there exists an edge $(v, v') \in E$ such that $(v', L_i(v) \cdot c(v, v'))$ is winning. Define $\nu(v, i)$ to be this v'.

We show that the strategy σ is winning. Consider a play $\pi = (v_0, v_1) \cdot (v_1, v_2) \cdots$ consistent with σ, and $i_0 \cdot i_1 \cdots$ the sequence of memory states assumed along this play. Denote π_k the prefix of π of length k, we prove that for all k, $L_{i_k}(v_k) \subseteq c(\pi_k)^{-1} W$. Note that by definition, $(v_k, L_{i_k}(v_k))$ is winning, so $L_{i_k}(v_k) \neq \emptyset$, implying that $c(\pi_k)^{-1} W \neq \emptyset$.

We proceed by induction. For $k = 0$, it follows from $L_1(v_0) \subseteq W$. Let $k > 0$, the induction hypothesis is $L_{i_{k-1}}(v_{k-1}) \subseteq c(\pi_{k-1})^{-1} W$. We distinguish two cases.

- Either v_{k-1} belongs to Eve, then by construction of σ we have that $(v_k, L_{i_{k-1}}(v_{k-1}) \cdot c(v_{k-1}, v_k))$ is winning. It follows that the update function is well defined, and $L_{i_k}(v_k) \subseteq L_{i_{k-1}}(v_{k-1}) \cdot c(v_{k-1}, v_k)$, which together with the induction hypothesis implies $L_{i_k}(v_k) \subseteq c(\pi_k)^{-1} W$.
- Or v_{k-1} belongs to Adam. Since Adam cannot escape $\mathcal{W}_E(\mathcal{G} \times \mathcal{M})$, we have that $(v_k, L_{i_{k-1}}(v_{k-1}) \cdot c(v_{k-1}, v_k))$ is winning, and the same reasoning concludes.

It follows that the strategy σ is winning, concluding the proof of Lemma 3. ◀

We now get rid of the regularity assumption. This means that for a vertex v, there may not be a minimal left quotient L such that (v, L) is winning. To get around this difficulty, the semantics of a memory state is not anymore a left quotient for each vertex, but rather a decreasing sequence of left quotients for each vertex.

Note that the proof of Lemma 4 uses the finite degree assumption, and is the only proof in the paper to do so. We will show in Section 4 that the result fails without this assumption.

▶ **Lemma 4** (Upper bound). *Let $\mathcal{G} = (\mathcal{A}, W)$ be a game with a safety condition W. If Eve wins, then she has a winning strategy using at most K memory states, where K is the width of* $(\text{Res}(W), \subseteq)$.

Consequently, for all safety conditions W, $\text{mem}(W)$ *is smaller than or equal to the width of* $(\text{Res}(W), \subseteq)$.

Proof. We use the same notations as for the proof of Lemma 3, and construct a memory structure together with a winning strategy using this memory structure.

Let K be the cardinal of the maximal antichain of left quotients of W. We construct the memory structure $\mathcal{M}^* = (\{1, \ldots, K\}, 1, \mu)$, and the strategy σ induced by the next-move function ν.

Let v be a vertex in $\mathcal{A}$. We consider the set $\mathcal{W}(v)$ of left quotients L such that (v, L) is winning. We split $\mathcal{W}(v)$ into maximal decreasing (finite or infinite) sequences of left quotients, denoted $\ell_1(v), \ldots, \ell_p(v)$, for some $p \leq K$. Furthermore, we choose $\ell_1(v_0)$ such that $W \in \ell_1(v_0)$. (Indeed, by assumption (v_0, W) is winning.)

We say that (v, ℓ) is winning if for all $L \in \ell$, we have that (v, L) is winning. For ℓ a sequence of left quotients and $a \in A$, we define $\ell \cdot a$ component-wise. Note that even if ℓ is infinite, it may be that $\ell \cdot a$ is finite.

We define the update function: $\mu(i, (v, v'))$ is a j as follows.

- If $\ell_i(v) \cdot c(v, v')$ is finite, denote $L \cdot c(v, v')$ its last element. Choose j such that $L \cdot c(v, v') \in \ell_j(v')$. Note that in general, such a j may not exist; it does exist if $(v', \ell_i(v) \cdot c(v, v'))$ is winning, and we will prove that this will always be the case when playing the strategy σ.
- If $\ell_i(v) \cdot c(v, v')$ is infinite, then choose j such that $\ell_j(v')$ has an infinite intersection with $\ell_i(v) \cdot c(v, v')$. Such a j exists without any assumption.

We define the next-move function ν (inducing σ). Let $v \in V_\exists$, and consider $(v, \ell_i(v))$. Let $L \in \ell_i(v)$, Eve wins from (v, L), so there exists an edge $(v, v') \in E$ such that $(v', L \cdot c(v, v'))$ is winning, we say that $(v, v') \in E$ is good for L. Since $\mathcal{W}(v')$ is upward closed, if $(v, v') \in E$ is good for L, then it is good for every L' such that $L \subseteq L'$. We argue that there exists an edge $(v, v') \in E$ that is good for all $L \in \ell_i(v)$, *i.e.* such that $(v', \ell_i(v) \cdot c(v, v'))$ is winning; define $\nu(v, i)$ to be this v'. There are two cases:

- Either $\ell_i(v)$ is finite, denote L its last element. Since $\ell_i(v)$ is decreasing, an edge good for L is good for all $L' \in \ell_i(v)$.
- Or $\ell_i(v)$ is infinite. The vertex v has finite degree, so there exists an edge which is good for infinitely many $L \in \ell_i(v)$. Since $\ell_i(v)$ is decreasing, it is good for all $L' \in \ell_i(v)$.

We show that the strategy σ is winning. Consider a play $\pi = (v_0, v_1) \cdot (v_1, v_2) \cdots$ consistent with σ, and $i_0 \cdot i_1 \cdots$ the sequence of memory states assumed along this play. Denote π_k the prefix of π of length k, we prove that for all k, there exists $L \in \ell_{i_k}(v_k)$ such that $L \subseteq c(\pi_k)^{-1}W$. Note that by definition, $(v_k, \ell_{i_k}(v_k))$ is winning, so $c(\pi_k)^{-1}W \neq \emptyset$.

We proceed by induction. For $k = 0$, it follows from $W \in \ell_1(v_0)$. Let $k > 0$, the induction hypothesis implies the existence of $L \in \ell_{i_{k-1}}(v_{k-1})$ such that $L \subseteq c(\pi_{k-1})^{-1}W$. We distinguish two cases, and denote $c_k = c(v_{k-1}, v_k)$.

- Either v_{k-1} belongs to Eve, then by construction of σ we have that $(v_k, \ell_{i_{k-1}}(v_{k-1}) \cdot c_k)$ is winning. It follows that the update function is well defined, and:

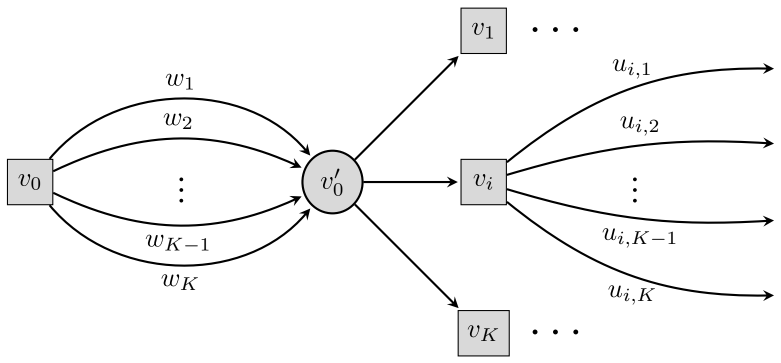


Figure 1 The lower bound.

1. If $\ell_{i_{k-1}}(v_{k-1}) \cdot c_k$ is finite, denote $L' \cdot c_k$ its last element, we have $L' \cdot c_k \in \ell_{i_k}(v_k)$. Since $L' \cdot c_k$ is the last element of $\ell_{i_{k-1}}(v_{k-1}) \cdot c_k$, it follows that $L' \subseteq L$. We have thus $L' \cdot c_k \subseteq L \cdot c_k$, so $L' \subseteq c(\pi_k)^{-1}W$, and $L' \cdot c_k \in \ell_{i_k}(v_k)$.
2. If $\ell_{i_{k-1}}(v_{k-1}) \cdot c_k$ is infinite, $\ell_{i_k}(v_k)$ has an infinite intersection with $\ell_{i_{k-1}}(v_{k-1}) \cdot c_k$. So there exists $L' \subseteq L$ with $L' \in \ell_{i_{k-1}}(v_{k-1})$ such that $L' \cdot c_k$ is in this intersection. We have $L' \cdot c_k \in \ell_{i_k}(v_k)$ and $L' \subseteq c(\pi_k)^{-1}W$.

- Or v_{k-1} belongs to Adam. Since Adam cannot escape $\mathcal{W}_E(\mathcal{G} \times \mathcal{M})$, we have that $(v_k, \ell_{i_{k-1}}(v_{k-1}) \cdot c_k)$ is winning, and the same reasoning concludes.

It follows that the strategy σ is winning, concluding the proof of Lemma 4. ◀

A Matching Lower Bound

▶ **Lemma 5** (Lower bound). *Let W be a safety condition. There exists a game $\mathcal{G} = (\mathcal{A}, W)$ where Eve wins but she has no winning strategy using less than K memory states where K is the width of* $(\mathrm{Res}(W), \subseteq)$.

Consequently, for all safety conditions W, $\mathrm{mem}(W)$ *is greater than or equal to the width of* $(\mathrm{Res}(W), \subseteq)$.

Proof. Consider $\{w_1^{-1}W, \dots, w_K^{-1}W\}$ an antichain of left quotients of W. For $i \neq j$, there exists $u_{i,j} \in A^\omega$ such that $u_{i,j} \in w_i^{-1}W$ and $u_{i,j} \notin w_j^{-1}W$.

We describe the game, illustrated in Figure 1. A play consists in three steps:

1. From v_0 to v'_0: Adam chooses a word in $\{w_1, \dots, w_K\}$;
2. Eve chooses between K options;
3. say Eve chose the i^{th} option, then Adam chooses between the $K-1$ words $u_{i,j}$ for $j \neq i$.

We first show that Eve has a winning strategy from v_0, using K memory states. It consists in choosing the i^{th} option whenever Adam chooses the word u_i: whatever Adam chooses at the third step, $w_i \cdot u_{i,j} \in W$.

We now show that there exists no winning strategy using less than K memory states. Indeed, such a strategy will not comply with the above strategy and for some $i \neq j$, choose the j^{th} option if Adam chooses w_i. Then Adam wins by playing $u_{i,j}$, since $w_i \cdot u_{i,j} \notin W$. ◀

Tight Bounds

Putting together upper and lower bounds, we proved the following result:

▶ **Theorem 6.** *For all safety conditions W,* $\mathrm{mem}(W)$ *is the width of* $(\mathrm{Res}(W), \subseteq)$.

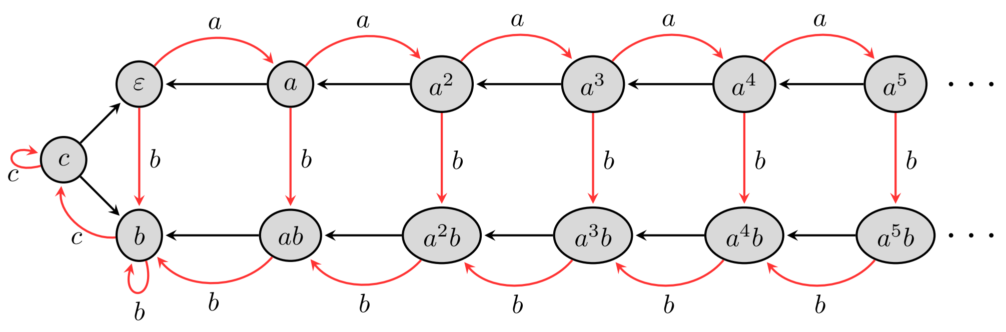


Figure 2 The outbidding condition: more b's than a's.

A condition W is half-positional if $\mathrm{mem}(W) = 1$. Characterizing half-positional conditions has been a fruitful topic over the last few years [8, 11]. In the case of safety conditions, we obtain the following characterization:

▶ **Corollary 7.** *For all safety conditions W, W is half-positional if and only if the inclusion is a linear order over* $\mathrm{Res}(W)$.

4 Examples and Applications

In this section, we instantiate Theorem 6 on different examples. We chose four examples:

- The outbidding condition shows the difference between graphs with finite degree and graphs with infinite degree; in particular, it gives a counter example to Lemma 4 when dropping the finite degree assumption,
- The energy condition is a non-regular half-positional safety condition,
- The generalized safety condition is a regular safety condition for which the partially ordered set of left quotients has a nice well-known combinatorical structure,
- The boundedness condition is a central piece in the theory of regular cost functions.

When representing the partial order $(\mathrm{Res}(W), \subseteq)$ for a given W, we use the following convention: a black edge from L to L' means that $L \subseteq L'$, and a red edge labeled a from L to L' means that $L' = L \cdot a$, so the red structure is the minimal (although possibly infinite) deterministic automaton recognizing W.

Outbidding Games

Let $A = \{a, b, c\}$ and $W = \{a^n \cdot b^p \cdot c^\omega \mid n \leq p\} \cup \{a^\omega\} \cup a^* \cdot b^\omega$. It is a non-regular safety condition, called the outbidding condition. Figure 2 represents the partial order $(\mathrm{Res}(W), \subseteq)$. Its width is three: there are two incomparable infinite increasing sequences of left quotients, $((a^n)^{-1}W)_{n\in\mathbb{N}}$ and $((b \cdot a^n)^{-1}W)_{n\in\mathbb{N}}$, and $c^{-1}W$.

Hence thanks to Theorem 6, $\mathrm{mem}(W) = 3$. However, there exists an outbidding game where Eve wins but needs infinite memory for this. This does not contradict Theorem 6, as this game, represented in Figure 3, has a vertex of infinite degree. It goes as follows: first Adam picks a number n, and then Eve takes over: she has to pick a number p, higher than or equal to n. A finite memory strategy can only choose from finitely many options, hence cannot win against all strategies of Adam.

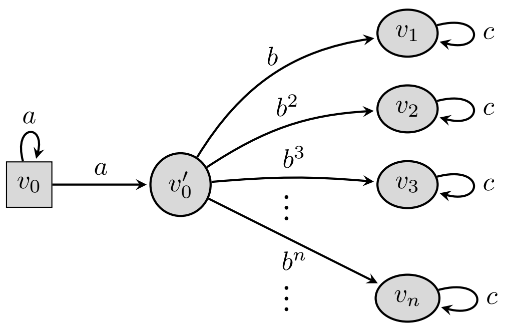


Figure 3 An outbidding game with infinite degree where Eve needs infinite memory to win.

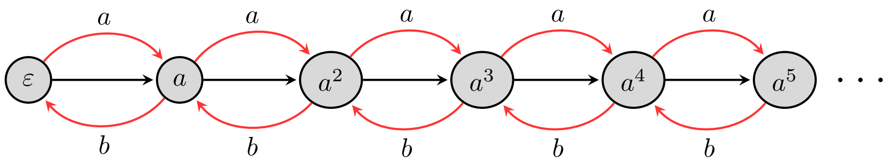


Figure 4 The energy condition: always more a's than b's.

Energy Games

The setup for the energy condition is the following: assume we are monitoring a resource. We denote by A the set of actions on this resource, which is any monotonic function $f : \mathbb{N} \to \mathbb{N}$, as for instance:

- consuming one unit of the resource,
- reloading by one unit,
- emptying the resource,
- consuming half of the current energy level.

Define the energy condition by

$$W = \{w \mid \text{ the energy level in } w \text{ remains always non-negative}\} .$$

It is a non-regular safety condition. Energy games and several variants have been extensively studied [1, 2, 3]. Figure 4 represents the partial order $(\text{Res}(W), \subseteq)$, with only two actions: a reloads by one unit, and b consumes one unit. In general, if the actions are monotonic, then the left quotients are totally ordered by inclusion, so thanks to Theorem 6 we have $\text{mem}(W) = 1$.

▶ **Corollary 8.** *The energy games are half-positional.*

Generalized Safety Games

This example originates from the study of generalized reachability games [6, 7]. A generalized reachability condition is a (finite) conjunction of reachability conditions. Here we take the opponent's vantage point: a generalized safety condition is a (finite) disjunction of (internal) safety conditions. Specifically, let $A = \{\perp, 1, \ldots, k\}$: each letter is a color, and $\perp$ is uncolored. Let $W = \{w = w_0 w_1 \cdots \mid \exists i \in \{1, \ldots, k\}, \forall n, w_n \neq i\}$, it is satisfied if at least one color is not seen along the play. It is a safety condition. Figure 4 represents the partial order $(\text{Res}(W), \subseteq)$ for $k = 3$. The left quotients are all the strict subsets of $\{1, \ldots, k\}$. The width of this partial order is $\binom{k}{\lfloor k/2 \rfloor}$, according to the well-known Sperner's Lemma from combinatorics.

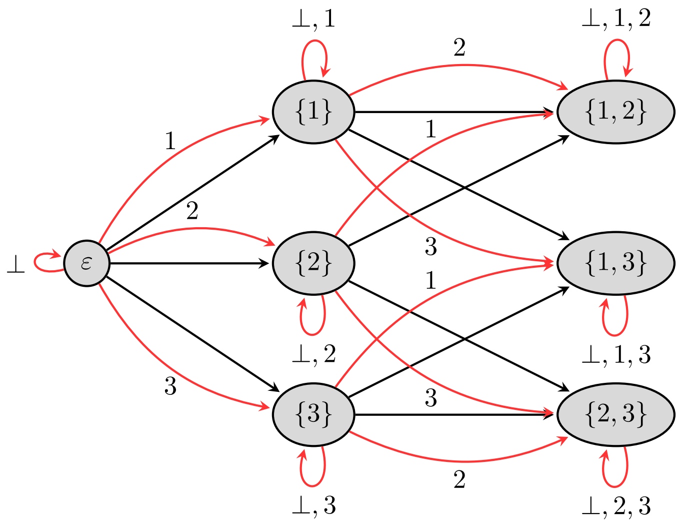


Figure 5 The generalized safety condition.

▶ **Corollary 9.** *For all generalized safety games with k colors, if Eve wins, then she has a winning strategy with $\binom{k}{\lfloor k/2 \rfloor}$ memory states.*

Furthermore, for all k, there exists a generalized safety game with k colors where Eve wins, but has no winning strategy using less than $\binom{k}{\lfloor k/2 \rfloor}$ memory states.

Games with Counters

This example originates from the theory of regular cost functions [4]. Let $N \in \mathbb{N}$, and define the boundedness condition W_N involving a counter as follows:

$$W_N = \{w \mid \text{the counter value in } w \text{ remains bounded by } N\} .$$

For the set of actions, we consider any monotonic action (even non-regular), *i.e.* function $f : \mathbb{N} \to \mathbb{N}$ such that if $i \leq j$ then $f(i) \leq f(j)$, as for instance:

- leaving the counter value unchanged,
- incrementing the counter value by one,
- resetting the counter value to zero,
- dividing the counter value by two, rounded down,
- increasing the counter value to the next power of two.

The condition W_N is a regular safety condition.

▶ **Corollary 10.** *The boundedness games are half-positional.*

5 Conclusion and Perspectives

We considered general safety conditions and characterized their memory requirements. Specifically, the memory requirements of a safety condition W is the width of the partially ordered set $(\text{Res}(W), \subseteq)$. This is the first general result characterizing the memory requirements for *some* non-regular conditions, based on their topological properties. We hope that this is the first stone on the path of memory requirements characterizations for much more conditions.

References

1 Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008.

2 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003.

3 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012.

4 Thomas Colcombet. Regular cost functions, part I: Logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.

5 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110. IEEE Computer Society, 1997.

6 Nathanaël Fijalkow and Florian Horn. The surprizing complexity of reachability games. *CoRR*, abs/1010.2420, 2010.

7 Nathanaël Fijalkow and Florian Horn. Les jeux d'accessibilité généralisée. *Technique et Science Informatiques*, 32(9-10):931–949, 2013.

8 Hugo Gimbert. *Jeux Positionnels*. PhD thesis, Université Paris 7, 2007.

9 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

10 Eryk Kopczyński. Personal communication.

11 Eryk Kopczyński. *Half-Positional Determinacy of Infinite Games*. PhD thesis, University of Warsaw, 2009.

12 Orna Kupferman. Variations on safety. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2014.

Metaconfluence of Calculi with Explicit Substitutions at a Distance*

Flávio L. C. de Moura†[1], Delia Kesner[2], and Mauricio Ayala-Rincón[1,3]

1 Departamento de Ciência da Computação, Universidade de Brasília
2 Université Paris-Diderot, SPC, PPS, CNRS
3 Departamento de Matemática, Universidade de Brasília

Abstract

Confluence is a key property of rewriting calculi that guarantees uniqueness of normal-forms when they exist. *Metaconfluence* is even more general, and guarantees confluence on *open/meta* terms, *i.e.* terms with holes, called *metavariables* that can be filled up with other (open/meta) terms. The difficulty to deal with open terms comes from the fact that the structure of metaterms is only *partially* known, so that some reduction rules became blocked by the metavariables. In this work, we establish metaconfluence for a family of calculi with explicit substitutions (ES) that enjoy preservation of strong-normalization (PSN) and that act *at a distance*. For that, we first extend the notion of reduction on metaterms in such a way that explicit substitutions are never structurally moved, *i.e.* they also act at a distance on metaterms. The resulting reduction relations are still rewriting systems, *i.e.* they do not include equational axioms, thus providing for the first time an interesting family of λ-calculi with explicit substitutions that enjoy both PSN and metaconfluence without requiring sophisticated notions of reduction modulo a set of equations.

1998 ACM Subject Classification F.4.1 Mathematical Logic: Lambda calculus and related systems

Keywords and phrases Confluence, Explicit Substitutions, Lambda Calculi

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.391

1 Introduction

Confluence is a key property of rewriting calculi that guarantees determinism of computations [9]. In confluent calculi, different reduction sequences starting at the same term always converge. When terms are enriched with *metavariables*, used to denote unknown parts of incomplete proofs/programs, we talk instead about *metaconfluence*, which in general does not follow directly from confluence.

In this paper we study metaconfluence of λ-calculi with explicit substitutions (ES), which are extensions of the λ-calculus being able to internalize the substitution operation [1, 20, 17, 13, 4]. Such calculi are used to refine/implement the notion of β-reduction in functional languages like Ocaml and Haskell, and proof-assistants like Coq, Isabelle, λProlog and PVS. Metaterms are notably introduced in this framework to implement higher-order unification and matching [16, 15, 8, 14] in proof-assistants. Indeed, a logical computational language based on higher-order resolution might be implemented using a calculus in which higher-order

* Work partially funded by the international project DeCOPA STIC-AmSud 146/2012.
† The first author was partially supported by FEMAT.

unification can be treated in a natural way through the use of metavariables. Surprisingly, metaconfluence does not follows directly from confluence. Indeed, when the ES operator is propagated w.r.t. the structure of (meta)terms, then metavariables naturally block such propagations. The problem can be illustrated in terms of the following (simplified) rewriting system (we assume no capture of free variables holds):

$$\begin{array}{lcllcl} (\lambda x.t)u & \mapsto & t[x/u] & (\lambda x.t)[y/u] & \mapsto & \lambda x.t[y/u] \\ (tv)[y/u] & \mapsto & t[y/u]v[y/u] & t[y/u] & \mapsto & t, \text{ if } y \text{ does not occur free in } t \end{array}$$

which generates the following diverging reduction sequences on metaterms:

$$t[y/u][x/v] \twoheadleftarrow ((\lambda x.t)v)[y/u] \to t[x/v][y/u]$$

where y does not occur free in v, $\to$ denotes the contextual closure of $\mapsto$ and $\twoheadrightarrow$ the reflexive-transitive closure of $\to$. Thus, there exist many λ-calculi with ES that are confluent but not metaconfluent. Some of them, as for example, $\lambda\sigma$ [1] and λs [19], were extended respectively to $\lambda\sigma_{\Uparrow}$ [12] and λs_e [20] in order to regain metaconfluence. Nevertheless, these extended calculi do not enjoy PSN [25, 18], thus showing the fragility of such rewriting systems.

Another solution was adopted by calculi with ES inspired from linear logic proof-nets [28], such as $\lambda\mathtt{es}$ [21] and $\lambda\mathtt{ex}$ [22]. Metaconfluence is recovered in these calculi by adding to the rewriting systems an equational axiom for commutation of independent substitutions:

$$t[y/u][x/v] \sim t[x/v][y/u]$$

where x (resp. y) does not occur free in u (resp. v). The resulting reduction calculi turned out to have very good properties, in particular, metaconfluence and PSN can live together [22]. However, equational reasoning becomes unavoidable, and even if commutation of independent substitutions is obtained for free when λ-terms are represented by proof-nets, this is not the case in classical implementations which use algebraic λ-terms.

In this paper we give a solution to this problem by pushing further the ES paradigm inspired from Linear-Logic Proof-Nets. In particular, there are nowadays several calculi, also inspired from Linear-Logic Proof-Nets, that are based on the idea that the ES operation acts *at a distance* and does not need to be percolated over the structure of terms. Typical examples of such calculi are the linear substitution (or Milner's) calculus and the structural lambda-calculus (see resp. [26] and [4]), that belong to this new paradigm and have been successfully used for different applications such as implicit complexity [7], the theory of abstract standardization [3], or abstract machines [2].

We prove metaconfluence for a family of calculi with ES that act *at a distance*, namely the substitution calculus [5], the linear substitution calculus [26] and the structural λ-calculus [4]. The resulting reduction systems are still simple rewriting systems, i.e. they do not include equational axioms. The key of our solution relies on a new notion of substitution that propagates/applies only those that are affecting real variables, by keeping fixed the ones affecting metavariables. For instance, if $\mathbb{Y}_x$ denotes a metavariable in a context having only the free variable x, then, according to our new notion of metasubstitution, the term $(\mathbb{Y}_x\ x)[x/u]$ reduces to $(\mathbb{Y}_x\ u)[x/u]$ and not to $\mathbb{Y}_x[x/u]u$ (which was the solution adopted in [24, 27]). The idea is that the *real* variable x of the metaterm $\mathbb{Y}_x\ x$ can be substituted by u, as usual, but the substitution $[x/u]$ remains fixed in its place and does not percolate the application, it must be delayed because of the metavariable $\mathbb{Y}_x$. This notion of metasubstitution turns out to be essential in the development of the results we show in this paper.

To establish metaconfluence for our three calculi, we use the Hindley-Rosen Theorem [10], which states that if two *confluent* rewriting systems *commute* then their *union is also* *confluent*. To do so, each rewriting system we treat in this paper is split into two rewriting systems, say $\mathcal{R}$ and §. We then show that $\mathcal{R}$ and § alone are confluent. Finally, we show that $\mathcal{R}$ *strongly commutes* with §, *i.e.* $\forall t_0, t_1, t_2$, if $t_0 \to_{\mathcal{R}} t_1$ and $t_0 \to_{\S} t_2$, $\exists t_3$ s.t. $t_1 \twoheadrightarrow_{\S} t_3$ and $t_2 \overset{=}{\to}_{\mathcal{R}} t_3$ (*i.e.* $t_2 \to_{\mathcal{R}} t_3$ or $t_2 = t_3$). Since strongly commutation implies commutation [9], then we are done by Hindley-Rosen Theorem.

We thus provide an interesting set of λ-calculi with ES that act at a distance, which enjoy both PSN and metaconfluence without requiring sophisticated notions of reduction modulo a set of equations. Moreover, in contrast to available proofs of metaconfluence in the literature [12, 20, 27, 24], which are non-trivial, our approach just needs a simple reasoning about commutation of reductions, a standard notion used in abstract reduction systems [11].

2 Common Syntax for Terms

Explicit substitutions calculi with names are built over a simple grammar which is an extension of that of the λ-calculus:

$$t, u ::= x \mid t\ u \mid \lambda x.t \mid t[x/u] \tag{1}$$

The symbol x is called a *variable*, $\lambda x.t$ an *abstraction*, $t\ u$ an *application* and $t[x/u]$ a term with an *explicit substitution (ES)* $[x/u]$, *i.e.* a substitution waiting to be applied. The abstraction $\lambda x.t$ and the ES $t[x/u]$ both bind x in t. The notions of **free** and **bound** variables are defined as usual, in particular, $\mathtt{fv}(t[x/u]) := \mathtt{fv}(t) \setminus \{x\} \cup \mathtt{fv}(u)$, $\mathtt{fv}(\lambda x.t) := \mathtt{fv}(t) \setminus \{x\}$, $\mathtt{bv}(t[x/u]) := \mathtt{bv}(t) \cup \{x\} \cup \mathtt{bv}(u)$ and $\mathtt{bv}(\lambda x.t) := \mathtt{bv}(t) \cup \{x\}$. We work with the standard notion of α-conversion *i.e.* the bound variables can be renamed in order to avoid clashes with the free ones. Thus, terms are always considered modulo α-equivalence, *i.e.* we work on α-equivalence classes of terms. We use $|t|_x$ to denote the number of free occurrences of the variable x in the term t. When $|t|_x = n \geq 2$, we write $t_{\langle x|y \rangle}$ for the **non-deterministic replacement** of i $(1 \leq i \leq n-1)$ free occurrences of x in t by a *fresh* variable y. Thus for example, given $u = (x\ z)[z/x]$, we have $|u|_x = 2$ so that only one replacement of x in u can be done to construct $u_{\langle x|y \rangle}$, which then denotes either $(y\ z)[z/x]$ or $(x\ z)[z/y]$ but not $(y\ z)[z/y]$. Contexts are defined as usual, *i.e.* they are given by the following grammar:

$$C ::= \Box \mid Ct \mid tC \mid \lambda x.C \mid C[x/t] \mid t[x/C]$$

We write $C[t]$ for the term obtained by replacing the hole $\Box$ of C by t, thus *e.g.* $(\Box y)[x] = xy$ and $(\lambda x.\Box)[x] = \lambda x.x$. We write $C[\![u]\!]$ when the free variables of u are not captured by the context C, thus for example, $C[\![x]\!]$ denotes the term xy if $C = \Box y$, and $\lambda y.x$, if $C = \lambda x.\Box$.

Substitutions are (finite) functions from variables to terms. We denote a non-empty substitution σ by $\{x_1/u_1, \ldots, x_n/u_n\}$ $(n \geq 1)$ and the empty substitution by $\mathtt{Id}$. The domain of the substitution σ is given by $\mathtt{dom}(\sigma) := \{x \mid \sigma(x) \neq x\}$. The set $\mathtt{var}(\sigma)$ is given by $\cup_{x \in \mathtt{dom}(\sigma)} \mathtt{fv}(\sigma(x))$. The **application** of a **substitution** σ to a term t is defined by induction on the structure of terms as follows:

$$\begin{array}{lllllll}
x\sigma & := & \sigma(x) & \text{if } x \in \mathtt{dom}(\sigma) & (\lambda y.t)\sigma & := & \lambda y.t\sigma \quad \text{if } y \notin \mathtt{var}(\sigma) \\
y\sigma & := & y & \text{if } y \notin \mathtt{dom}(\sigma) & t[y/u]\sigma & := & t\sigma[y/u\sigma] \quad \text{if } y \notin \mathtt{var}(\sigma) \\
(tu)\sigma & := & (t\sigma)(u\sigma) & & & &
\end{array}$$

Here it should be stressed that the third and fourth rules are conceived modulo α-conversion. Thus for example $(\lambda y.x)\{x/y\} = \lambda z.y$. Remark that $t\{x/u\} = t$ if $x \notin \mathtt{fv}(t)$.

We now present the reduction rules of three calculi with ES acting at a distance that are based on grammar (1). The first one, known as the substitution calculus, splits the non-terminating β-rule of the λ-calculus into two terminating rules $\mathtt{dB}$ and $\mathtt{s}$ (see Fig. 1).

$(\lambda x.t)\mathtt{L}\ u$	$\mapsto_{\mathtt{dB}}$	$t[x/u]\mathtt{L}$
$t[x/u]$	$\mapsto_{\mathtt{s}}$	$t\{x/u\}$

Figure 1 The substitution calculus for terms.

$(\lambda x.t)\mathtt{L}\ u$	$\mapsto_{\mathtt{dB}}$	$t[x/u]\mathtt{L}$	
$\mathtt{C}[\![x]\!][x/u]$	$\mapsto_{\mathtt{ls}}$	$\mathtt{C}[\![u]\!][x/u]$	
$t[x/u]$	$\mapsto_{\mathtt{w}}$	t	if $\lvert t\rvert_x = 0$

Figure 2 The linear substitution calculus for terms.

$(\lambda x.t)\mathtt{L}\ u$	$\mapsto_{\mathtt{dB}}$	$t[x/u]\mathtt{L}$	
$t[x/u]$	$\mapsto_{\mathtt{c}}$	$t_{\langle x\lvert y\rangle}[x/u][y/u]$	if $\lvert t\rvert_x > 1$
$t[x/u]$	$\mapsto_{\mathtt{d}}$	$t\{x/u\}$	if $\lvert t\rvert_x = 1$
$t[x/u]$	$\mapsto_{\mathtt{w}}$	t	if $\lvert t\rvert_x = 0$

Figure 3 The structural substitution calculus for terms.

The $\mathtt{L}$ symbol appearing in the $\mathtt{dB}$-rule, also called $\mathtt{d}$*istant* $\mathtt{B}$*eta*, denotes a (possibly empty) list of substitutions of the form $[x_1/t_1][x_2/t_2]\dots[x_n/t_n]$ $(n \geq 0)$ [1] The resulting reduction relation, obtained by the contextual closure of the rewriting rules, is written $\rightarrow_{\lambda_{\mathtt{sub}}}$.

The second calculus (see Fig. 2) is known as the linear substitution calculus. Rule $\mapsto_{\mathtt{dB}}$ (resp. $\mapsto_{\mathtt{ls}}$) comes from the structural λ-calculus [4] (resp. Milner's calculus [26]), while $\mapsto_{\mathtt{w}}$ belongs to both calculi. The calculus performs *partial* substitution in the sense that only one free variable occurrence is substituted at a time. This partial substitution, performed by means of the $\mathtt{ls}$-rule (for $\mathtt{l}$inear $\mathtt{s}$ubstitution), is non-deterministic, *i.e.* $\mathtt{ls}$ randomly chooses the free variable occurrence of x to be substituted by u. The resulting reduction relation, obtained by the contextual closure of the rewriting rules, is written $\rightarrow_{\lambda_{\mathtt{lsub}}}$.

The third calculus (see Fig. 3) is the structural λ-calculus [4]. The $\mathtt{dB}$-rule triggers computation. The $\mathtt{c}$-rule duplicates ES which affect a term $t_{\langle x|y\rangle}$, denoting, as defined above, some non-deterministic replacement of a non-empty subset of the free occurrences of the variable x in t by a fresh variable y. Metasubstitution on variables is only performed by the $\mathtt{d}$-rule where the variables have only one single occurrence in the term. The resulting reduction relation, obtained by the contextual closure of the rewriting rules, is written $\rightarrow_{\lambda_{\mathtt{str}}}$.

In what follows we denote by $\rightarrow_r$ (resp. $\twoheadrightarrow_r$) the contextual (resp. reflexive-transitive) closure of each rewriting rule $\mapsto_r$ (resp. reduction relation $\rightarrow_r$), for $r \in \{\mathtt{dB}, \mathtt{s}, \mathtt{ls}, \mathtt{w}, \mathtt{c}, \mathtt{d}\}$ introduced before. For each calculus, the reduction relation associated to its substitution calculus, *i.e.* generated by all its rewriting rules except $\mapsto_{\mathtt{dB}}$, are defined by $\rightarrow_{\mathtt{s}} := \rightarrow_{\lambda_{\mathtt{sub}}} \setminus \rightarrow_{\mathtt{dB}}$, $\rightarrow_{\mathtt{lsub}} := \rightarrow_{\lambda_{\mathtt{lsub}}} \setminus \rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{str}} := \rightarrow_{\lambda_{\mathtt{str}}} \setminus \rightarrow_{\mathtt{dB}}$ respectively. A reduction relation $\rightarrow_{\mathcal{R}}$ is said to be **confluent** on terms (resp. metaterms) iff for all terms (resp. metaterms) t_0, t_1, t_2, if $t_0 \twoheadrightarrow_{\mathcal{R}} t_1$ and $t_0 \twoheadrightarrow_{\mathcal{R}} t_2$, there exists a term (resp. metaterm) t_3 s.t. $t_1 \twoheadrightarrow_{\mathcal{R}} t_3$ and $t_2 \twoheadrightarrow_{\mathcal{R}} t_3$.

Here are some examples of reduction sequences from the term $t = (\lambda x.xx)y$:

In $\lambda_{\mathtt{sub}}$: $t \rightarrow_{\mathtt{dB}} (xx)[x/y] \rightarrow_{\mathtt{s}} yy$

In $\lambda_{\mathtt{lsub}}$: $t \rightarrow_{\mathtt{dB}} (xx)[x/y] \rightarrow_{\mathtt{ls}} (xy)[x/y] \rightarrow_{\mathtt{ls}} (yy)[x/y] \rightarrow_{\mathtt{w}} yy$

In $\lambda_{\mathtt{str}}$: $t \rightarrow_{\mathtt{dB}} (xx)[x/y] \rightarrow_{\mathtt{c}} (xx')[x/y][x'/y] \rightarrow_{\mathtt{d}} (yx')[x'/y] \rightarrow_{\mathtt{d}} yy$

[1] Formally, the list $\mathtt{L}$ is a context generated by the grammar $\Box \mid \mathtt{L}[x/t]$.

All the calculi presented above, let us write $\mathcal{R}$, enjoy good properties, specially: (*simulation*) every β-reduction step in the λ-calculus can be performed in $\mathcal{R}$, (*confluence*) all divergent reduction sequences in $\mathcal{R}$ can be closed and (*preservation of β-strong normalization*) every β-strongly normalizing λ-term is also $\mathcal{R}$-strongly normalizing.

3 Common Syntax for MetaTerms

We now extend the grammar of terms to metaterms by adding metavariables which denote incomplete proofs/programs in higher-order theories. In particular, metavariables are used in higher-order unification to denote unkown partial solutions to be instantiated by the unification procedure [15, 8, 14]. We use $\mathbb{X}_\Delta$ to denote a metavariable with free variables in the set Δ. The grammar (1) introduced in Sec. 2 is then extended as follows:

$$t, u ::= x \mid \mathbb{X}_\Delta \mid t\ u \mid \lambda x.t \mid t[x/u] \qquad (2)$$

We also extend the notation $|t|_x$ to metaterms, thus *e.g.* $|(y\mathbb{X}_y)[z/\mathbb{Z}_y]|_y = 3$. We distinguish between **free meta** and **real** variables. They are both defined by induction as follows.

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\mathtt{fm}(x) & := & \emptyset & \mathtt{fr}(x) & := & \{x\} \\
\mathtt{fm}(\mathbb{X}_\Delta) & := & \Delta & \mathtt{fr}(\mathbb{X}_\Delta) & := & \emptyset \\
\mathtt{fm}(tu) & := & \mathtt{fm}(t) \cup \mathtt{fm}(u) & \mathtt{fr}(tu) & := & \mathtt{fr}(t) \cup \mathtt{fr}(u) \\
\mathtt{fm}(\lambda x.t) & := & \mathtt{fm}(t) \setminus \{x\} & \mathtt{fr}(\lambda x.t) & := & \mathtt{fr}(t) \setminus \{x\} \\
\mathtt{fm}(t[x/u]) & := & \mathtt{fm}(t) \setminus \{x\} \cup \mathtt{fm}(u) & \mathtt{fr}(t[x/u]) & := & \mathtt{fr}(t) \setminus \{x\} \cup \mathtt{fr}(u)
\end{array}$$

Thus for example, given $t = (\mathbb{X}_{\{x,y\}} z)[x/\mathbb{Y}_\emptyset]$ we have $\mathtt{fm}(t) = \{y\}$ and $\mathtt{fr}(t) = \{z\}$. The set of **free** variables of a metaterm is given by $\mathtt{fv}(t) := \mathtt{fm}(t) \cup \mathtt{fr}(t)$. We extend the non-deterministic operation $\__{\langle \_ | \_ \rangle}$ introduced in Sec. 2 to metaterms as expected.

For each λ-calculus in this paper, the metaconfluence proof uses the termination property of the corresponding substitution subcalculus. The first substitution calculus, given by the reduction relation $\to_{\mathtt{s}}$, is trivially terminating. In order to prove termination of the substitution subcalculus $\to_{\mathtt{lsub}} := \to_{\mathtt{ls}} \cup \to_{\mathtt{w}}$ we use a decreasing measure based on the notion of multiplicity [24]. Indeed, the **size** of a metaterm is recursively defined as follows:

$$\begin{array}{lcl@{\qquad}lcl}
\mathtt{sz}(x) = \mathtt{sz}(\mathbb{X}_\Delta) & := & 1 & \mathtt{sz}(t\ u) & := & \mathtt{sz}(t) + \mathtt{sz}(u) \\
\mathtt{sz}(\lambda x.t) & := & \mathtt{sz}(t) & \mathtt{sz}(t[x/u]) & := & \mathtt{sz}(t) + \mathtt{sz}(u) \cdot (1 + \mathtt{ml}_x(t))
\end{array}$$

where $\mathtt{ml}_x(t)$, the **multiplicity** of the variable x in the metaterm t, is defined by:

$$\mathtt{ml}_x(t) := 0 \qquad \text{if } x \notin \mathtt{fv}(t), \text{ otherwise}$$

$$\begin{array}{lcl@{\qquad}lcl}
\mathtt{ml}_x(x) = \mathtt{ml}_x(\mathbb{X}_\Delta) & := & 1 & \mathtt{ml}_x(t\ u) & := & \mathtt{ml}_x(t) + \mathtt{ml}_x(u) \\
\mathtt{ml}_x(\lambda y.t) & := & \mathtt{ml}_x(t) & \mathtt{ml}_x(t[y/u]) & := & \mathtt{ml}_x(t) + \mathtt{ml}_x(u) \cdot (1 + \mathtt{ml}_y(t))
\end{array}$$

We have for example $\mathtt{sz}((x\ x)[x/\lambda y.y]) = 5$, $\mathtt{sz}((x\ z)[x/\lambda y.y][z/\lambda y.y]) = 6$ and $\mathtt{sz}((z\ z)[z/x\ x][x/\lambda y.y]) = 15$. Observe that $\mathtt{sz}(t) \geq 1$ and $\mathtt{ml}_x(t) \geq 0$. Moreover, $x \notin \mathtt{fv}(t)$ implies $\mathtt{ml}_x(t) = 0$. It is easy to extend these measures to contexts by adding $\mathtt{sz}(\Box) = 0$ and $\mathtt{ml}_x(\Box) = 0$.

While this measure is decreasing for $\mathtt{lsub}$, *i.e.* $t \to_{\mathtt{lsub}} t'$ implies $\mathtt{sz}(t) > \mathtt{sz}(t')$ (see Sec. 4.1 for details), this is not the case for the subcalculus $\to_{\mathtt{str}} := \to_{\mathtt{c}} \cup \to_{\mathtt{d}} \cup \to_{\mathtt{w}}$. Thus for example, $(x\ x)[x/u] \to_{\mathtt{c}} (x\ z)[x/u][z/u]$ but $\mathtt{sz}((x\ x)[x/u]) < \mathtt{sz}((x\ z)[x/u][z/u])$. We then introduce another measure (cf. [4]) which will be used in Sec. 4.3 to show that $\to_{\mathtt{str}}$

terminates. In what follows $[\,]$ denotes the empty multiset, $\sqcup$ the multiset union and $n \cdot [a_1, \ldots, a_n]$ the multiset $[n \cdot a_1, \ldots, n \cdot a_n]$.

The **multimeasure** of a metaterm t, written $\mathtt{jm}(t)$, is a multiset of integers defined as:

$$\begin{array}{lllll} \mathtt{jm}(x) = \mathtt{jm}(\mathbb{X}_\Delta) & := & [\,] & \mathtt{jm}(t\ u) & := & \mathtt{jm}(t) \sqcup \mathtt{jm}(u) \\ \mathtt{jm}(\lambda x.t) & := & \mathtt{jm}(t) & \mathtt{jm}(t[x/u]) & := & [\mathtt{P}_x(t)] \sqcup \mathtt{jm}(t) \sqcup \max(1, \mathtt{P}_x(t)) \cdot \mathtt{jm}(u) \end{array}$$

where $\mathtt{P}_x(t)$ denotes the **potential multiplicity** of the variable x in the term t and is recursively defined on α-equivalence classes of t as follows: $\mathtt{P}_x(t) := 0$ if $x \notin \mathtt{fv}(t)$, otherwise

$$\begin{array}{llllll} \mathtt{P}_x(x) = \mathtt{P}_x(\mathbb{X}_\Delta) & := & 1 & \mathtt{P}_x(t\ u) & := & \mathtt{P}_x(t) + \mathtt{P}_x(u) \\ \mathtt{P}_x(\lambda y.t) & := & \mathtt{P}_x(t) & \mathtt{P}_x(t[y/u]) & := & \mathtt{P}_x(t) + \max(1, \mathtt{P}_y(t)) \cdot \mathtt{P}_x(u) \end{array}$$

Note that in the second case, necessarily $x \in \Delta$. Thus for example, $\mathtt{jm}((x\ x)[x/\lambda y.y]) = [2]$; $\mathtt{jm}((x\ z)[x/\lambda y.y][z/\lambda y.y]) = [1,1]$ and $\mathtt{jm}((z\ z)[z/x\ x][x/\lambda y.y]) = [4,2]$.

4 Metaconfluence

This section is devoted to the proofs of metaconfluence of our three calculi. We start by extending the notion of metasubstitution introduced in Sec. 2 to metaterms. A first approach, already used in [23] for the $\lambda\mathtt{ex}$-calculus, is obtained by adding to the metasubstitution operation on terms the following case :

$$\mathbb{X}_\Delta\{x/u\} = \begin{cases} \mathbb{X}_\Delta[x/u] & \text{if } x \in \Delta \\ \mathbb{X}_\Delta & \text{otherwise.} \end{cases} \tag{3}$$

Nevertheless, if one naively uses this specification to extend the $\mathtt{s}$-rule to metaterms, termination is lost as the following example shows: $\mathbb{X}_{\{x\}}[x/u] \to_{\mathtt{s}} \mathbb{X}_{\{x\}}\{x/u\} = \mathbb{X}_{\{x\}}[x/u] \to_{\mathtt{s}} \ldots$ This can be recovered by simply restricting the form of the metaterms t on the left-hand side of the $\mathtt{s}$-rule to those that are not metavariables affected by ES (as done for example in [24]). However, even with this restriction, confluence fails:

$$X_{\{x,y\}}[x/u][y/v] \twoheadleftarrow (\lambda y.\mathbb{X}_{\{x,y\}})[x/u]\ v \to_{\mathtt{dB}} \mathbb{X}_{\{x,y\}}[y/v][x/u]$$

One can then add equations between metaterms to allow permutation of independent substitutions (cf. [21]) in order to close this divergent diagram. But then equational reasoning becomes necessary to deal with the resulting reduction system (a reduction system modulo); this could be particularly problematic from an implementation point of view.

In this paper we present another approach where no additional equations are necessary to guarantee metaconfluence. We start by extending the notion of metasubstitution to metaterms as follows. The (capture-free) **fixed metasubstitution** of x by the metaterm u in the metaterm t, written $t[\![x/u]\!]$, is given by:

$$t[\![x/u]\!] := \begin{cases} t\{\!\{x/u\}\!\}[x/u], & \text{if } x \in \mathtt{fm}(t) \\ t\{\!\{x/u\}\!\}, & \text{if } x \notin \mathtt{fm}(t) \end{cases}$$

where the operation $\_\{\!\{\_/\_\}\!\}$ is defined as follows: $t\{\!\{x/u\}\!\} := t$ if $x \notin \mathtt{fr}(t)$, otherwise

$$\begin{array}{lll} x\{\!\{x/u\}\!\} := u; & (\lambda y.v)\{\!\{x/u\}\!\} := \lambda y.v\{\!\{x/u\}\!\} & (x \neq y\ \&\ y \notin \mathtt{fv}(u)); \\ (tv)\{\!\{x/u\}\!\} := t\{\!\{x/u\}\!\} v\{\!\{x/u\}\!\}; & t[y/v]\{\!\{x/u\}\!\} := t\{\!\{x/u\}\!\}[y/v\{\!\{x/u\}\!\}] & (x \neq y\ \&\ y \notin \mathtt{fv}(u)). \end{array}$$

Thus for example, $(\lambda y.x\mathbb{X}_{\{x\}})[\![x/y]\!] = (\lambda z.y\mathbb{X}_{\{x\}})[x/y]$. Remark that renaming of the bound variable y was done to avoid capture of free variables. The *real* free occurrence of x

$$\boxed{\begin{array}{lll l}(\lambda x.t)\mathtt{L}\ u & \mapsto_{\mathtt{dB}} & t[x/u]\mathtt{L} & \\ t[x/u] & \mapsto_{\mathtt{s}} & t\llbracket x/u\rrbracket & \text{if } x \notin \mathtt{fm}(t) \text{ or } x \in \mathtt{fr}(t)\end{array}}$$

Figure 4 The substitution calculus for metaterms.

$$\boxed{\begin{array}{lll l}(\lambda x.t)\mathtt{L}\ u & \mapsto_{\mathtt{dB}} & t[x/u]\mathtt{L} & \\ t[x/u] & \mapsto_{\mathtt{c}} & t_{\langle x|y\rangle}[x/u][y/u] & \text{if } |t|_x > 1 \\ t[x/u] & \mapsto_{\mathtt{d}} & t\llbracket x/u\rrbracket & \text{if } |t|_x = 1 \text{ and } x \notin \mathtt{fm}(t) \\ t[x/u] & \mapsto_{\mathtt{w}} & t & \text{if } |t|_x = 0\end{array}}$$

Figure 5 The structural lambda calculus for metaterms.

was substituted by y, however, the ES $[x/y]$ remains fixed in the resulting term since it is affecting a metavariable with scope x.

The above definition is a key notion of this work: it is able to capture metasubstitution on *terms*, but it is compatible with *metaterms*. Formally, the metasubstitution $\_\llbracket\_/\_\rrbracket$ is splited into two complementary notions of substitution: the ES $\_[\_/\_]$, which is fixed whenever there is a metavariable with scope in the domain of this substitution; and the implicit substitution $\_\{\!\{\_/\_\}\!\}$ on terms, that only acts on real variables.

We can now reformulate the first and the third λ-calculi presented before by using fixed metasubstitution $\llbracket x/u\rrbracket$ on *metaterms* instead of $\{x/u\}$ on *terms*. The resulting reduction relations are shown in Fig. 4 and Fig. 5, respectively. In the case of the linear substitution calculus, since the reduction relation on terms does not use metasubstitution, we can keep exactly the same rewriting rules in Fig. 2 to specify reduction on *metaterms*. In the three cases, the resulting reduction systems on metaterms are conservative w.r.t. their respective reduction notions on terms.

4.1 The Substitution Calculus enjoys MetaConfluence

This section presents the metaconfluence proof for the substitution calculus. We start by stating some useful properties concerning the notion of metasubstitution that will be important in the rest of this section.

▶ **Lemma 1.** *Let t, u, v be metaterms. If $x \neq y$ and $x \notin \mathtt{fv}(v)$ then $t\{\!\{x/u\}\!\}\{\!\{y/v\}\!\} = t\{\!\{y/v\}\!\}\{\!\{x/u\{\!\{y/v\}\!\}\}\!\}$ and $t\llbracket x/u\rrbracket\{\!\{y/v\}\!\} = t\{\!\{y/v\}\!\}\llbracket x/u\{\!\{y/v\}\!\}\rrbracket$.*

Proof. The first statement is by induction on t and the second one uses the first one. ◀

The next lemma states that the substitution calculus on metaterms is stable w.r.t. the new notion of metasubstitution.

▶ **Lemma 2** (Stability). *Let t, u be metaterms. Let $r \in \{\mathtt{dB}, \mathtt{s}\}$.*
- *If $t \to_r t'$, then $t\{\!\{x/u\}\!\} \to_r t'\{\!\{x/u\}\!\}$ and $t\llbracket x/u\rrbracket \to_r t'\llbracket x/u\rrbracket$.*
- *If $u \to_r u'$, then $t\{\!\{x/u\}\!\} \twoheadrightarrow_r t\{\!\{x/u'\}\!\}$ and $t\llbracket x/u\rrbracket \twoheadrightarrow_r t\llbracket x/u'\rrbracket$.*

Proof.
- The first statement is by induction on $t \to_{\lambda_{\mathtt{sub}}} t'$ using Lem. 1. The second one uses the first one.
- The first statement is by induction on $u \to_{\lambda_{\mathtt{sub}}} u'$ and the second one by using the first one. ◀

The stability properties are necessary to prove that the reduction systems $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{s}}$ strongly commutes.

▶ **Theorem 3** (Strong Commutation). *$\forall t_0, t_1, t_2$, if $t_0 \rightarrow_{\mathtt{s}} t_1$ and $t_0 \rightarrow_{\mathtt{dB}} t_2$, $\exists t_3$ s.t. $t_1 \twoheadrightarrow_{\mathtt{dB}} t_3$ and $t_2 \overset{=}{\rightarrow}_{\mathtt{s}} t_3$.*

Proof. By induction on the reduction relation. We only show here the key cases:

$$\begin{tikzcd}
t[x/u] \arrow[r, "\mathtt{s}"] \arrow[d, "\mathtt{dB}"] & t\llbracket x/u\rrbracket \arrow[d, "\mathtt{dB}\ (Lem.\ 2)"] \\
t'[x/u] \arrow[r, "\mathtt{s}"] & t'\llbracket x/u\rrbracket
\end{tikzcd}
\quad
\begin{tikzcd}
t[x/u] \arrow[r, "\mathtt{s}"] \arrow[d, "\mathtt{dB}"] & t\llbracket x/u\rrbracket \arrow[d, two heads, "\mathtt{dB}(Lem.\ 2)"] \\
t[x/u'] \arrow[r, "\mathtt{s}"] & t\llbracket x/u'\rrbracket
\end{tikzcd}
\quad
\begin{tikzcd}
(\lambda x.t)\mathtt{L}u \arrow[r, "\mathtt{s}"] \arrow[d, "\mathtt{dB}"] & (\lambda x.t')\mathtt{L}u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L} \arrow[r, "\mathtt{s}"] & t'[x/u]\mathtt{L}
\end{tikzcd}$$

$$\begin{tikzcd}
(\lambda x.t)\mathtt{L}u \arrow[r, "\mathtt{s}"] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}'u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L} \arrow[r, "\mathtt{s}"] & t[x/u]\mathtt{L}'
\end{tikzcd}
\quad
\begin{tikzcd}
(\lambda x.t)\mathtt{L}u \arrow[r, "\mathtt{s}"] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}u' \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L} \arrow[r, "\mathtt{s}"] & t[x/u']\mathtt{L}
\end{tikzcd}
\quad
\begin{tikzcd}
(\lambda x.t)\mathtt{L}_1[y/v]\mathtt{L}_2 u \arrow[r, "\mathtt{s}"] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}_1\llbracket y/v\rrbracket\mathtt{L}_2 u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L}_1[y/v]\mathtt{L}_2 \arrow[r, "\mathtt{s}"] & t[x/u]\mathtt{L}_1\llbracket y/v\rrbracket\mathtt{L}_2
\end{tikzcd}$$

◀

We can now conclude metaconfluence of the substitution calculus as follows:

▶ **Corollary 4.** *The reduction relation $\rightarrow_{\lambda_{\mathtt{sub}}}$ is confluent on metaterms.*

Proof. Let $\rightarrow_{\lambda_{\mathtt{sub}}} := \rightarrow_{\mathtt{dB}} \cup \rightarrow_{\mathtt{s}}$. Both $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{str}}$ are trivially confluent. They commute (Theorem 3). We conclude by the Hindley-Rosen Theorem [10] introduced in Sec 1. ◀

4.2 The Linear Substitution Calculus enjoys MetaConfluence

In this section we prove metaconfluence for the linear substitution calculus. As in the previous section, we first prove that the systems $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{lsub}}$ strongly commute, where, as defined in Sec. 3 we have $\rightarrow_{\mathtt{lsub}} := \rightarrow_{\mathtt{ls}} \cup \rightarrow_{\mathtt{w}}$.

▶ **Theorem 5** (Strong Commutation). *$\forall t_0, t_1, t_2$, if $t_0 \rightarrow_{\mathtt{lsub}} t_1$ and $t_0 \rightarrow_{\mathtt{dB}} t_2$, $\exists t_3$ s.t. $t_1 \twoheadrightarrow_{\mathtt{dB}} t_3$ and $t_2 \overset{=}{\rightarrow}_{\mathtt{lsub}} t_3$.*

Proof. By induction on the reduction relations, then, for the base cases, by case analysis of overlapping local divergences. Most of the cases are straightforward, we only show here the more interesting ones.

$$\begin{tikzcd}
\mathtt{C}\llbracket x\rrbracket[x/u] \arrow[r, "\mathtt{ls}"] \arrow[d, "\mathtt{dB}"] & \mathtt{C}\llbracket u\rrbracket[x/u] \arrow[d, "\mathtt{dB}"] \\
\mathtt{C}'\llbracket x\rrbracket[x/u] \arrow[r, "\mathtt{ls}"] & \mathtt{C}'\llbracket u\rrbracket[x/u]
\end{tikzcd}
\quad
\begin{tikzcd}
(\lambda x.\mathtt{C}\llbracket y\rrbracket)\mathtt{L}_1[y/v]\mathtt{L}_2 u \arrow[r, "\mathtt{ls}"] \arrow[d, "\mathtt{dB}"] & (\lambda x.\mathtt{C}\llbracket v\rrbracket)\mathtt{L}_1[y/v]\mathtt{L}_2 u \arrow[d, "\mathtt{dB}"] \\
\mathtt{C}\llbracket y\rrbracket[x/u]\mathtt{L}_1[y/v]\mathtt{L}_2 \arrow[r, "\mathtt{ls}"] & \mathtt{C}\llbracket v\rrbracket[x/u]\mathtt{L}_1[y/v]\mathtt{L}_2
\end{tikzcd}$$

$$\begin{tikzcd}
\mathtt{C}\llbracket x\rrbracket[x/u] \arrow[r, "\mathtt{ls}"] \arrow[d, "\mathtt{dB}"] & \mathtt{C}\llbracket u\rrbracket[x/u] \arrow[d, two heads, "\mathtt{dB}"] \\
\mathtt{C}\llbracket x\rrbracket[x/u'] \arrow[r, "\mathtt{ls}"] & \mathtt{C}\llbracket u'\rrbracket[x/u']
\end{tikzcd}
\quad
\begin{tikzcd}
(\lambda x.t)\mathtt{L}_1[y/v]\mathtt{L}_2 u \arrow[r, "\mathtt{w}"] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}_1\mathtt{L}_2 u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L}_1[y/v]\mathtt{L}_2 \arrow[r, "\mathtt{w}"] & t[x/u]\mathtt{L}_1\mathtt{L}_2
\end{tikzcd}$$

$$\begin{tikzcd}
(\lambda x.t)\mathtt{L}_{11}[z/\mathtt{C}\llbracket y\rrbracket]\mathtt{L}_{12}[y/v]\mathtt{L}_2 u \arrow[r, "\mathtt{ls}"] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}_{11}[z/\mathtt{C}\llbracket v\rrbracket]\mathtt{L}_{12}[y/v]\mathtt{L}_2 u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L}_{11}[z/\mathtt{C}\llbracket y\rrbracket]\mathtt{L}_{12}[y/v]\mathtt{L}_2 \arrow[r, "\mathtt{ls}"] & t[x/u]\mathtt{L}_{11}[z/\mathtt{C}\llbracket v\rrbracket]\mathtt{L}_{12}[y/v]\mathtt{L}_2
\end{tikzcd}$$

◀

The next goal is to establish confluence of the subsystem $\rightarrow_{\mathtt{lsub}}$. For that we first need to establish termination, that can be proved using some intermediate auxiliary results.

▶ **Lemma 6.** *Let $x \neq y$ and assume $y \notin \mathtt{fv}(v)$. Then* $\mathtt{ml}_x(\mathtt{C}[\![y]\!]) + \mathtt{ml}_y(\mathtt{C}[\![y]\!]) \cdot \mathtt{ml}_x(v) = \mathtt{ml}_x(\mathtt{C}[\![v]\!]) + \mathtt{ml}_y(\mathtt{C}[\![v]\!]) \cdot \mathtt{ml}_x(v)$.

Proof. By induction on $\mathtt{C}$. ◀

The following lemma states compatibility of $\rightarrow_{\mathtt{lsub}}$ w.r.t $\mathtt{ml}_x(\cdot)$.

▶ **Lemma 7** (Compatibility). *For all x, t and t', such that $t \rightarrow_{\mathtt{lsub}} t'$, $\mathtt{ml}_x(t) \geq \mathtt{ml}_x(t')$.*

Proof. The proof is by induction on the reduction relation. We show the base cases, since the inductive cases are straightfoward:
Case $\mathtt{C}[\![z]\!][z/u] \rightarrow_{\mathtt{ls}} \mathtt{C}[\![u]\!][z/u]$: $\mathtt{ml}_x(\mathtt{C}[\![z]\!][z/u]) \geq \mathtt{ml}_x(\mathtt{C}[\![u]\!][z/u])$ if and only if $\mathtt{ml}_x(\mathtt{C}[\![z]\!]) + \mathtt{ml}_z(\mathtt{C}[\![z]\!]) \cdot \mathtt{ml}_x(u) \geq \mathtt{ml}_x(\mathtt{C}[\![u]\!]) + \mathtt{ml}_z(\mathtt{C}[\![u]\!]) \cdot \mathtt{ml}_x(u)$, which holds by Lem. 6, since $z \notin \mathtt{fv}(u)$. Actually, in this case the equality holds.
Case $t[z/u] \rightarrow_{\mathtt{w}} t$: $\mathtt{ml}_x(t[z/u]) = \mathtt{ml}_x(t) + \mathtt{ml}_x(u) + \mathtt{ml}_z(t) \cdot \mathtt{ml}_x(u) \geq \mathtt{ml}_x(t)$. ◀

▶ **Lemma 8.** *Let $x \neq y$ such that $x, y \notin \mathtt{fv}(v)$. Then $\mathtt{ml}_x(C[\![x]\!]) > \mathtt{ml}_x(C[\![v]\!])$ and $\mathtt{ml}_y(C[\![x]\!]) = \mathtt{ml}_y(C[\![v]\!])$.*

Proof. The proof is by simultaneous induction on C. ◀

▶ **Lemma 9.** *The system $\rightarrow_{\mathtt{lsub}}$ is terminating on metaterms.*

Proof. We show that $t \rightarrow_{\mathtt{lsub}} t'$ implies $\mathtt{sz}(t) > \mathtt{sz}(t')$ so that $\rightarrow_{\mathtt{lsub}}$ is necessarily terminating. The proof is by induction on $t \rightarrow_{\mathtt{lsub}} t'$ and uses Lem. 7 and Lem. 8. ◀

▶ **Lemma 10.** *The reduction relations $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{lsub}}$ are confluent.*

Proof. As noticed before the relation $\rightarrow_{\mathtt{dB}}$ is trivially confluent. Since $\rightarrow_{\mathtt{lsub}}$ is terminating on metaterms (Lem. 9), in order to conclude confluence it is enough to verify joinability of all critical pairs. The sole critical peak is built overlapping the $\mathtt{ls}$-rule with itself: $\mathtt{C}[\![u]\!][x/u]\ {}_{\mathtt{ls}}\!\leftarrow \mathtt{C}[\![x]\!][x/u] = t[x/u] = \mathtt{C}'[\![x]\!][x/u] \rightarrow_{\mathtt{ls}} \mathtt{C}'[\![u]\!][x/u]$. In other words, t can be written as $\mathtt{D}'[\![x, x]\!]$, where $\mathtt{D}'$ is a two-hole context. Then, the critical peak is joinable: $\mathtt{D}'[\![u, x]\!][x/u] \rightarrow_{\mathtt{ls}} \mathtt{D}'[\![u, u]\!][x/u]\ {}_{\mathtt{ls}}\!\leftarrow \mathtt{D}'[\![x, u]\!][x/u]$. ◀

We can now conclude metaconfluence for the linear substitution calculus as follows:

▶ **Corollary 11.** *The reduction relation $\rightarrow_{\lambda_{\mathtt{lsub}}}$ is confluent on metaterms.*

Proof. Let $\rightarrow_{\lambda_{\mathtt{lsub}}} := \rightarrow_{\mathtt{dB}} \cup \rightarrow_{\mathtt{lsub}}$. Since both $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{lsub}}$ are confluent (Lem 10) and strongly commute (Theorem 5), their union is confluent using the Hindley-Rosen Theorem [10] introduced in Sec. 1. ◀

4.3 The Structural Lambda Calculus enjoys MetaConfluence

In this section we prove metaconfluence for the structural lambda calculus. As in the previous section, we first prove that the systems $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{str}}$ strongly commute, where, as defined in Sec. 3, we have $\rightarrow_{\mathtt{str}} := \rightarrow_{\mathtt{c}} \cup \rightarrow_{\mathtt{d}} \cup \rightarrow_{\mathtt{w}}$.

▶ **Theorem 12** (Strong Commutation). *$\forall t_0, t_1, t_2$, if $t_0 \rightarrow_{\mathtt{str}} t_1$ and $t_0 \rightarrow_{\mathtt{dB}} t_2$, $\exists t_3$ s.t. $t_1 \twoheadrightarrow_{\mathtt{dB}} t_3$ and $t_2 \overset{=}{\rightarrow}_{\mathtt{str}} t_3$.*

Proof. By induction on the reduction relations. We only show here the diagrams of the more interesting cases.

$$\begin{tikzcd}
(\lambda x.t)\mathtt{L}_1[y/v]\mathtt{L}_2 u \arrow[r, "\mathtt{w}"'] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}_1\mathtt{L}_2 u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L}_1[y/v]\mathtt{L}_2 \arrow[r, "\mathtt{w}"'] & t[x/u]\mathtt{L}_1\mathtt{L}_2
\end{tikzcd}
\qquad
\begin{tikzcd}
(\lambda x.t)\mathtt{L}_1[y/v]\mathtt{L}_2 u \arrow[r, "\mathtt{d}"'] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}_1[\![y/v]\!]\mathtt{L}_2 u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L}_1[y/v]\mathtt{L}_2 \arrow[r, "\mathtt{d}"'] & t[x/u]\mathtt{L}_1[\![y/v]\!]\mathtt{L}_2
\end{tikzcd}$$

$$\begin{tikzcd}
t[x/u] \arrow[r, "\mathtt{c}"'] \arrow[d, "\mathtt{dB}"] & t_{\langle y|x\rangle}[x/u][y/u] \arrow[d, two heads, "\mathtt{dB}"] \\
t[x/u'] \arrow[r, "\mathtt{c}"'] & t_{\langle y|x\rangle}[x/u'][y/u']
\end{tikzcd}
\qquad
\begin{tikzcd}
t[x/u] \arrow[r, "\mathtt{c}"'] \arrow[d, "\mathtt{dB}"] & t_{\langle y|x\rangle}[x/u][y/u] \arrow[d, "\mathtt{dB}"] \\
t'[x/u] \arrow[r, "\mathtt{c}"'] & t'_{\langle y|x\rangle}[x/u][y/u]
\end{tikzcd}$$

$$\begin{tikzcd}
(\lambda x.t)\mathtt{L}_1[y/v]\mathtt{L}_2 u \arrow[r, "\mathtt{c}"'] \arrow[d, "\mathtt{dB}"] & (\lambda x.t)\mathtt{L}_{1\langle y|y'\rangle}[y/v][y'/v]\mathtt{L}_2 u \arrow[d, "\mathtt{dB}"] \\
t[x/u]\mathtt{L}_1[y/v]\mathtt{L}_2 \arrow[r, "\mathtt{c}"'] & t[x/u]\mathtt{L}_{1\langle y|y'\rangle}[y/v][y'/v]\mathtt{L}_2
\end{tikzcd}$$

The subtle point here is that in the last two diagrams we need to choose the replacement $t'_{\langle y|x\rangle}$ (resp. $(t[x/u]\mathtt{L}_1)_{\langle y|y'\rangle}$) according to that we used for $t_{\langle y|x\rangle}$ (resp. $((\lambda x.t)\mathtt{L}_1)_{\langle y|y'\rangle}$). ◀

The next goal is to establish confluence of the subsystem $\rightarrow_{\mathtt{str}}$. For that we first need to establish termination, that can be proved using some intermediate auxiliary results.

▶ **Lemma 13.** *If $x \notin \mathtt{fv}(u)$ and $x \neq y$, then $\mathtt{P}_x(t) = \mathtt{P}_x(t[\![y/u]\!])$.*

Proof. The hypothesis implies $\mathtt{P}_x(u) = 0$. Moreover, we can easily prove by induction on t that $\mathtt{P}_x(t\{\!\{y/u\}\!\}) \overset{*}{=} \mathtt{P}_x(t)$, if $x \notin \mathtt{fv}(u)$ and $x \neq y$. We then consider all the possible cases for $t[\![y/u]\!]$.

1. If $y \in \mathtt{fm}(t)$ and $y \in \mathtt{fr}(t)$, then $\mathtt{P}_x(t[\![y/u]\!]) = \mathtt{P}_x(t\{\!\{y/u\}\!\}[y/u]) = \mathtt{P}_x(t\{\!\{y/u\}\!\}) \overset{*}{=} \mathtt{P}_x(t)$;
2. If $y \notin \mathtt{fm}(t)$ and $y \in \mathtt{fr}(t)$, then $\mathtt{P}_x(t[\![y/u]\!]) = \mathtt{P}_x(t\{\!\{y/u\}\!\}) \overset{*}{=} \mathtt{P}_x(t)$;
3. If $y \in \mathtt{fm}(t)$ and $y \notin \mathtt{fr}(t)$, then $\mathtt{P}_x(t[\![y/u]\!]) = \mathtt{P}_x(t[y/u]) = \mathtt{P}_x(t\{\!\{y/u\}\!\}) \overset{*}{=} \mathtt{P}_x(t)$;
4. If $y \notin \mathtt{fv}(t)$, then $\mathtt{P}_x(t[\![y/u]\!]) = \mathtt{P}_x(t)$. ◀

The following properties hold for metaterms. The proofs can be done by simple structural induction, where the metavariable case is straightforward and the other cases are in [6].

▶ **Lemma 14.** *Let t be a metaterm. Then*
1. *$|t|_x \leq \mathtt{P}_x(t)$.*
2. *If $x \notin \mathtt{fv}(u)$ then $\mathtt{P}_x(t) = \mathtt{P}_x(t[y/u])$.*
3. *If x, y, z are pairwise distinct and $z \notin \mathtt{fv}(t)$ then $\mathtt{P}_x(t) = \mathtt{P}_x(t_{\langle y|z\rangle})$.*
4. *If $y \notin \mathtt{fv}(t)$ and $t' = t_{\langle x|y\rangle}$ then $\mathtt{P}_x(t) = \mathtt{P}_x(t') + \mathtt{P}_y(t')$.*

▶ **Lemma 15.** *If $|t|_y = 1$ then $\mathtt{P}_x(t\{\!\{y/u\}\!\}) \leq \mathtt{P}_x(t) + \mathtt{P}_y(t) \cdot \mathtt{P}_x(u)$.*

Proof. By induction on t. ◀

▶ **Lemma 16.** *If $|t|_x = 1$ and $x \notin \mathtt{fm}(t)$ then $\mathtt{jm}(t[x/u]) \sqsupset \mathtt{jm}(t[\![x/u]\!])$.*

Proof. By induction on t using Lem. 13 and Lem. 14. ◀

▶ **Lemma 17.** *Let t, t' be metaterms. If $t \rightarrow_{\mathtt{str}} t'$ then $\mathtt{P}_x(t) \geq \mathtt{P}_x(t')$.*

Proof. By induction on $t \rightarrow_{\mathtt{str}} t'$ using Lem. 15 and Lem. 14. ◀

▶ **Lemma 18.** *The reduction relation $\rightarrow_{\mathtt{str}}$ is terminating on metaterms.*

Proof. By induction on $t \rightarrow_{\mathtt{str}} t'$ using Lem. 16 and Lem. 17. ◀

▶ **Lemma 19.** *The reduction relations $\rightarrow_{\mathtt{str}}$ and $\rightarrow_{\mathtt{dB}}$ are confluent.*

Proof. The reduction relation $\rightarrow_{\mathtt{str}}$ is terminating on metaterms (Lem. 18), and do not have critical pairs because its rules are mutually exclusive. Therefore, $\rightarrow_{\mathtt{str}}$ is confluent. ◀

Metaconfluence of the structural substitution calculus is then obtained as follows:

▶ **Corollary 20.** *The reduction relation $\rightarrow_{\lambda_{\mathtt{str}}}$ is confluent on metaterms.*

Proof. Let $\rightarrow_{\lambda_{\mathtt{str}}} := \rightarrow_{\mathtt{dB}} \cup \rightarrow_{\mathtt{str}}$. Both $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{str}}$ are confluent (Lem. 19) and strongly commute (Theorem 12), therefore their union is confluent by the Hindley-Rosen Theorem [10] introduced in Sec. 1. ◀

5 Conclusion

We define reduction for metaterms for three calculi with explicit substitutions that act at a distance, namely the substitution calculus, the linear substitution calculus and the structural lambda calculus. This is done by defining a subtle notion of metasubstitution, which is completely fixed for metavariables. In contrast to other specifications of λ-calculi with ES on metaterms, our resulting reduction systems do not contain equations, so that their equational theories are simple enough to be treated with simple rewriting techniques. In particular, our proofs of (meta)confluence can be achieved by using the well-known Hindley-Rosen Theorem.

As mentioned before, metaconfluence is an essential property of calculi with ES used to implement higher-order unification (HOU) procedures [15, 8]. Indeed, such algorithms need to compare typed metaterms in (η-long) normal form, which are unique by metaconfluence. They then generate new metavariables in order to denote partial solutions that need again to be in (η-long) normal form in order to recursively apply the algorithm. As future work, we want to investigate higher-order unification (HOU) procedures based on calculi acting at a distance. We believe that the simplicity and applicability of such calculi can lead to unification procedures that are simpler than already known unification procedures based on other ES calculi [15, 8].

References

1 M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

2 B. Accattoli, P. Barenbaum, and D. Mazza. Distilling abstract machines. Available at `https://sites.google.com/site/beniaminoaccattoli/`, 2014.

3 B. Accattoli, E. Bonelli, D. Kesner, and C. Lombardi. A nonstandard standardization theorem. In 41st Symposium on Principles of Programming Languages (POPL), editor, *ACM SIGPLAN-SIGACT*, pages 659–670, 2014.

4 B. Accattoli and D. Kesner. The structural lambda-calculus. In *19th EACSL Annual Conference on Computer Science and Logic (CSL)*, volume 6247 of *LNCS*, pages 381–395. Springer-Verlag, 2010.

5 B. Accattoli and D. Kesner. The permutative lambda calculus. In *LPAR*, pages 23–36, 2012.

6 B. Accattoli and D. Kesner. Preservation of strong normalisation modulo permutations for the structural lambda-calculus. *Logical Methods in Computer Science*, 8(1), 2012.

7 B. Accattoli and U. Dal Lago. Beta reduction is invariant, indeed. Accepted to LICS/CSL 2014.

8 M. Ayala-Rincón and F. Kamareddine. Unification via the λs_e-Style of Explicit Substitution. *The Logical Journal of the IGPL*, 9(4):489–523, 2001.

9 F. Baader and T. Nipkow. *Term Rewriting and* All That. Cambridge University Press, 1998.

10 H. Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

11 M. Bezem, J. W. Klop, and R. de Vrijer, editors. *Term Rewriting Seminar – Terese*. Cambridge University Press, 2003.

12 P.-L. Curien, T. Hardin, and J.-J. Levy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43:43–2, 1996.

13 R. David and B. Guillaume. A lambda-calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11(1):169–206, 2001.

14 F. L. C. de Moura, M. Ayala-Rincón, and F. Kamareddine. Higher-Order Unification: A structural relation between Huet's method and the one based on explicit substitutions. *Journal of Applied Logic*, 6(1):72–108, 2008.

15 G. Dowek, T. Hardin, and C. Kirchner. Higher order unification via explicit substitutions. *Inf. Comput.*, 157(1-2):183–235, 2000.

16 G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In *JICSLP*, pages 259–273, 1996.

17 Z. el A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a Calculus of Explicit Substitutions which Preserves Strong Normalization. *JFP*, 6(5):699–722, 1996.

18 B. Guillaume. The λs_e-calculus Does Not Preserve Strong Normalization. *J. of Func. Programming*, 10(4):321–325, 2000.

19 F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with Explicit Substitutions. In *Proc. of PLILP'95*, volume 982 of *LNCS*, pages 45–62. Springer, 1995.

20 F. Kamareddine and A. Ríos. Extending a λ-calculus with Explicit Substitution which Preserves Strong Normalisation into a Confluent Calculus on Open Terms. *Journal of Functional Programming*, 7:395–420, 1997.

21 D. Kesner. The theory of calculi with explicit substitutions revisited. In *CSL*, pages 238–252, 2007.

22 D. Kesner. Perpetuality for full and safe composition (in a constructive setting). In *To appear in Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008), Track B*, Reykjavik, Iceland,, 2008.

23 D. Kesner. A Theory of Explicit Substitutions with Safe and Full Composition. *Logical Methods in Computer Science*, 5(3:1):1–29, 2009.

24 D. Kesner and S. Ó Conchúir. Milner's Lambda Calculus with Partial Substitutions. Technical Report, Université Paris Diderot, 2008.

25 P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In *Proceedings of TLCA'95*, volume 902 of *LNCS*. Springer-Verlag, 1995.

26 R. Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). *ENTCS*, 175(3):65–73, 2007.

27 F. Renaud. *Les Ressources Explicites vues par la Théorie de la Réécriture*. PhD thesis, Université Paris Diderot - Paris 7, 2011. Available at `www.lix.polytechnique.fr/~renaud/these.pdf`.

28 Jean yves Girard. Proof-nets: The parallel syntax for proof-theory. In *Logic and Algebra*, pages 97–124. Marcel Dekker, 1996.

Behavioral Metrics via Functor Lifting

Paolo Baldan[1], Filippo Bonchi[2], Henning Kerstan[3], and Barbara König[3]

1 **Dipartimento di Matematica, Università di Padova, Italy**
`baldan@math.unipd.it`
2 **CNRS, ENS Lyon, Université de Lyon, France**
`filippo.bonchi@ens-lyon.fr`
3 **Universität Duisburg-Essen, Germany**
`henning.kerstan@uni-due.de`, `barbara_koenig@uni-due.de`

Abstract

We study behavioral metrics in an abstract coalgebraic setting. Given a coalgebra $\alpha\colon X \to FX$ in **Set**, where the functor F specifies the branching type, we define a framework for deriving pseudometrics on X which measure the behavioral distance of states.

A first crucial step is the lifting of the functor F on **Set** to a functor $\overline{F}$ in the category **PMet** of pseudometric spaces. We present two different approaches which can be viewed as generalizations of the Kantorovich and Wasserstein pseudometrics for probability measures. We show that the pseudometrics provided by the two approaches coincide on several natural examples, but in general they differ.

Then a final coalgebra for F in **Set** can be endowed with a behavioral distance resulting as the smallest solution of a fixed-point equation, yielding the final $\overline{F}$-coalgebra in **PMet**. The same technique, applied to an arbitrary coalgebra $\alpha\colon X \to FX$ in **Set**, provides the behavioral distance on X. Under some constraints we can prove that two states are at distance 0 if and only if they are behaviorally equivalent.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, D.2.4 Software/Program Verification

Keywords and phrases behavioral metric, functor lifting, pseudometric, coalgebra

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.403

1 Introduction

Increasingly, modelling formalisms are equipped with quantitative information, such as probability, time or weight. Such quantitative information should be taken into account when reasoning about behavioral equivalence of system states, such as bisimilarity. In this setting the appropriate notion is not necessarily equivalence, but a behavioral metric that measures the distance of the behavior of two states. In a quantitative setting, it is often unreasonable to assume that two states have exactly the same behavior, but it makes sense to express that their behavior differs by some (small) value ε.

The above considerations led to the study of behavioral metrics which aims at quantifying the the distance between the behavior of states. Since different states can have exactly the same behavior it is quite natural to consider *pseudometrics*, which allow different elements to be at zero distance.

Earlier contributions defined behavioral metrics in the setting of probabilistic systems [9, 23] and of metric transition systems [6]. Our aim is to generalize these ideas and to study behavioral metrics in a general coalgebraic setting. The theory of coalgebra [17] is nowadays a well-established tool for defining and reasoning about various state based transition systems such as deterministic, nondeterministic, weighted or probabilistic automata. Hence, it is the appropriate setting to ask and answer general questions about behavioral metrics.

- *How can we define behavioral metrics for transition systems with different branching types?* We provide a coalgebraic framework in the category of pseudometric spaces **PMet** that allows to define and reason about such metrics.
- *Are the behavioral metrics canonical in some way?* We provide a natural way to define metrics by lifting functors from **Set** to the category of pseudometric spaces. In fact, we study two liftings: the Kantorovich and the Wasserstein lifting and observe that they coincide in many cases. This provides us with a notion of canonicity and justification for the choice of metrics.
- *Does the measurement of distances affect behavioral equivalence?* If we start by considering coalgebras in **PMet** (as, e.g., in [23]), it is not entirely clear a priori whether the richer categorical structure influences the notion of behavioral equivalence. In our setting we start with coalgebras in **Set** and put distance measurements "on top", showing that, under some mild constraints, the original notion of behavioral equivalence is not compromised, in the sense that two states are behaviorally equivalent iff their distance is 0.
- *Are there generic algorithms to compute metrics?* Coalgebra is a valuable tool to define generic methods that can be instantiated to concrete cases in order to obtain prototype algorithms. In our case we give a (high-level) procedure for computing behavioral distances on a given coalgebra, based on determining the smallest solution of a fixed-point equation.

A central contribution of this paper is the lifting of a functor F from **Set** to **PMet**. Given a pseudometric space (X, d), the goal is to define a suitable pseudometric on FX. Such liftings of metrics have been extensively studied in transportation theory [24], e.g. for the case of the (discrete) probability distribution functor, which comes with a nice analogy: assume several cities (with fixed distances between them) and two probability distributions s, t on cities, representing supply and demand (in units of mass). The distance between s, t can be measured in two ways: the first is to set up an optimal transportation plan with minimal costs (in the following also called coupling) to transport goods from cities with excess supply to cities with excess demand. The cost of transport is determined by the product of mass and distance. In this way we obtain the Wasserstein distance. A different view is to imagine a logistics firm that is commissioned to handle the transport. It sets prices for each city and buys and sells for this price at every location. However, it has to ensure that the price function is nonexpansive, i.e., the difference of prices between two cities is smaller than the distance of the cities, otherwise it will not be worthwhile to outsource this task. This firm will attempt to maximize its profit, which can be considered as the Kantorovich distance of s, t. The Kantorovich-Rubinstein duality informs us that these two views lead to the exactly same result, a very good argument for the canonicity of this notion of distance.

It is our observation that these two notions of distance lifting can analogously be defined for arbitrary functors F, leading to a rich general theory. The lifting has an evaluation function as parameter. As concrete examples, besides the probability distribution functor, we study the (finite) powerset functor (resulting in the Hausdorff metric) and the coproduct and product bifunctors. In the case of the product bifunctor we consider different evaluation functions, each leading to a well-known product metric. The Kantorovich-Rubinstein duality holds for these functors, but it does not hold in general (we provide a counterexample).

After discussing functor liftings, we define coalgebraic behavioral pseudometrics and answer the questions above. Specifically we show how to compute distances on the final coalgebra as well as on arbitrary coalgebras via fixed-point iteration and we prove that the pseudometric obtained on the final coalgebra is indeed a metric. In [3] we discuss a fibrational perspective on our work and we compare with [13]. All proofs for our results are in [3].

2 Preliminaries, Notation and Evaluation Functions

We assume that the reader is familiar with the basic notions of category theory, especially with the definitions of functor, product, coproduct and weak pullbacks.

For a function $f\colon X \to Y$ and sets $A \subseteq X$, $B \subseteq Y$ we write $f[A] := \{f(a) \mid a \in A\}$ for the *image* of A and $f^{-1}[B] = \{a \in A \mid f(x) \in B\}$ for the *preimage* of B. If $Y \subseteq [0, \infty]$ and $f, g\colon X \to Y$ are functions we write $f \le g$ when $\forall x \in X : f(x) \le g(x)$.

Given a natural number $n \in \mathbb{N}$ and a family $(X_i)_{i=1}^n$ of sets X_i we denote the projections of the (cartesian) product of the X_i by $\pi_i^n\colon \prod_{i=1}^n X_i \to X_i$, or just by π_i if n is clear from the context. For a source $(f_i\colon X \to X_i)_{i=1}^n$ we denote the unique mediating arrow to the product by $\langle f_1, \ldots, f_n\rangle\colon X \to \prod_{i=1}^n X_i$. Similarly, given a family of arrows $(f_i\colon X_i \to Y_i)_{i=1}^n$, we write $f_1 \times \cdots \times f_n = \langle f_1 \circ \pi_1, \ldots, f_n \circ \pi_n\rangle\colon \prod_{i=1}^n X_i \to \prod_{i=1}^n Y_i$.

We quickly recap the basic ideas of coalgebras. Let F be an endofunctor on the category **Set** of sets and functions. An F-coalgebra is just a function $\alpha\colon X \to FX$. Given another F-coalgebra $\beta\colon Y \to FY$ a coalgebra homomorphism from α to β is a function $f\colon A \to B$ such that $\beta \circ f = Ff \circ \alpha$. We call an F-coalgebra $\kappa\colon \Omega \to F\Omega$ *final* if for any other coalgebra $\alpha\colon X \to FX$ there is a unique coalgebra homomorphism $[\![\_]\!]\colon X \to \Omega$. The final coalgebra need not exist but if it does it is unique up to isomorphism. It can be considered as the universe of all possible behaviors. If we have an endofunctor F such that a final coalgebra $\kappa\colon \Omega \to F\Omega$ exists then for any coalgebra $\alpha\colon X \to FX$ two states $x_1, x_2 \in X$ are said to be *behaviorally equivalent* if and only if $[\![x_1]\!] = [\![x_2]\!]$.

We now introduce some preliminaries about (pseudo)metric spaces. Our (pseudo)metrics assume values in a closed interval $[0, \top]$, where $\top \in (0, \infty]$ is a fixed maximal element (for our examples we will use $\top = 1$ or $\top = \infty$). In this way the set of (pseudo)metrics over a fixed set with pointwise order is a complete lattice (since $[0, \top]$ is) and the resulting category of pseudometric spaces is complete and cocomplete.

▶ **Definition 2.1** (Pseudometric, Pseudometric Space)**.** Given a set X, a *pseudometric on* X is a function $d\colon X \times X \to [0, \top]$ such that for all $x, y, z \in X$, the following axioms hold: $d(x, x) = 0$ (*reflexivity*), $d(x, y) = d(y, x)$ (*symmetry*), $d(x, z) \le d(x, y) + d(y, z)$ (*triangle inequality*). If additionally $d(x, y) = 0$ implies $x = y$, d is called a *metric*. A *(pseudo)metric space* is a pair (X, d) where X is a set and d is a (pseudo)metric on X.

By $d_e\colon [0, \top]^2 \to [0, \top]$ we denote the ordinary Euclidean distance on $[0, \top]$, i.e., $d_e(x, y) = |x - y|$ for $x, y \in [0, \top] \setminus \{\infty\}$, and – where appropriate – $d_e(x, \infty) = \infty$ if $x \ne \infty$ and $d_e(\infty, \infty) = 0$. Addition is defined in the usual way, in particular $x + \infty = \infty$ for $x \in [0, \infty]$.

Hereafter, we only consider those functions between pseudometric spaces that do not increase distances.

▶ **Definition 2.2** (Nonexpansive Function, Isometry)**.** Let (X, d_X), (Y, d_Y) be pseudometric spaces. A function $f\colon X \to Y$ is called *nonexpansive* if $d_Y \circ (f \times f) \le d_X$. In this case we write $f\colon (X, d_X) \xrightarrow{1} (Y, d_Y)$. If equality holds, f is called an *isometry*.

For our purposes it will turn out to be useful to consider the following alternative characterization of the triangle inequality using the concept of nonexpansive functions.

▶ **Lemma 2.3.** *A reflexive and symmetric function $d\colon X^2 \to [0,\top]$ satisfies the triangle inequality iff for all $x \in X$ the function $d(x,\_)\colon X \to [0,\top]$ is nonexpansive.*

As stated before, our definition of a pseudometric gives rise to a suitably rich category.

▶ **Definition 2.4** (Category of Pseudometric Spaces)**.** For a fixed $\top \in (0,\infty]$ we denote by $\mathbf{PMet}$ the category of all pseudometric spaces and nonexpansive functions.

This category is complete and cocomplete (see [3]) and, in particular, it has products and coproducts as we will see in Examples 5.1 and 5.2. We now introduce two motivating examples borrowed from [23] and [6].

▶ **Example 2.5** (Probabilistic Transition Systems and Behavioral Distance)**.** We regard probabilistic transition systems as coalgebras of the form $\alpha\colon X \to \mathcal{D}(X+\mathbf{1})$, where $\mathcal{D}$ is the probability distribution functor (with finite support) which maps a set X to the set $\mathcal{D}X = \{P\colon X \to [0,1] \mid \sum_{x\in X} P(x) = 1, P \text{ has finite support}\}$ and a function $f\colon X \to Y$ to the function $\mathcal{D}f\colon \mathcal{D}X \to \mathcal{D}Y, P \mapsto \lambda y.\sum_{x \in f^{-1}[\{y\}]} P(x)$. Here $\alpha(x)(y)$, for $x,y \in X$, denotes the probability of a transition from a state x to y and $\alpha(x)(\checkmark)$ stands for the probability of terminating from x (we use $\checkmark$ for the single element of the set $\mathbf{1}$).

In [23] a metric for the continuous version of these systems is introduced, by considering a discount factor $c \in (0,1)$. In the discrete case we obtain the behavioral distance $d\colon X^2 \to [0,1]$, defined as the least solution of the equation $d(x,y) = \overline{d}(\alpha(x),\alpha(y))$, where $x,y \in X$ and $\overline{d}\colon (\mathcal{D}(X+\mathbf{1}))^2 \to [0,1]$ is defined in two steps: First, $\hat{d}\colon (X+\mathbf{1})^2 \to [0,1]$ is defined as $\hat{d}(x,y) = c\cdot d(x,y)$ if $x,y \in X$, $\hat{d}(\checkmark,\checkmark) = 0$ and 1 otherwise. Then, for all $P_1,P_2 \in \mathcal{D}(X+\mathbf{1})$, $\overline{d}(P_1,P_2)$ is defined as the supremum of all values $\sum_{x\in X+\mathbf{1}} f(x)\cdot\big(P_1(x) - P_2(x)\big)$, with $f\colon (X+\mathbf{1},\hat{d}) \xrightarrow{1} ([0,1],d_e)$ being an arbitrary nonexpansive function. As we will further discuss in Example 3.3, $\overline{d}$ is the Kantorovich pseudometric given by the space $(X+\mathbf{1},\hat{d})$.

We consider a concrete example from [23], illustrated on the left of Figure 1. The behavioral distance of u and z is $d(u,z) = 1$ and hence $d(x,y) = c\cdot\varepsilon$.

▶ **Example 2.6** (Metric Transition Systems and Propositional Distances)**.** We give another example based on the notions of [6]. A finite set $\Sigma = \{r_1,\ldots,r_n\}$ of propositions is given and each proposition $r \in \Sigma$ is associated with a pseudometric space (M_r,d_r). A valuation u is a function with domain Σ that assigns to each $r \in \Sigma$ an element of M_r. We denote the set of all valuations by $\mathcal{U}[\Sigma]$. A metric transition system is a tuple $M = (S,\tau,\Sigma,[\cdot])$ with a set S of states, a transition relation $\tau \subseteq S \times S$, a finite set Σ of propositions and a valuation $[s]$ for each state $s \in S$. We write $\tau(s)$ for $\{s' \in S \mid (s,s') \in \tau\}$ and require that $\tau(s)$ is finite.

In [6] the propositional distance between two valuations is given by $\overline{pd}(u,v) = \max_{r\in\Sigma} d_r(u(r),v(r))$ for $u,v \in \mathcal{U}[\Sigma]$. The (undirected) branching distance $d\colon S\times S \to \mathbb{R}_0^+$ is defined as the smallest fixed-point of the following equation, where $s,t \in S$:

$$d(s,t) = \max\left\{\overline{pd}([s],[t]), \max_{s'\in\tau(s)}\min_{t'\in\tau(t)} d(s',t'), \max_{t'\in\tau(t)}\min_{s'\in\tau(s)} d(s',t')\right\} \tag{1}$$

Note that, apart from the first argument, this coincides with the Hausdorff distance.

We consider an example which appears similarly in [6] (see Figure 1, right) with a single proposition $r \in \Sigma$, where $M_r = [0,1]$ is equipped with the Euclidean distance d_e. According to (1), $d(x_1,y_1)$ equals the Hausdorff distance of the reals associated with the sets of successors, which is 0.3 (since this is the maximal distance of any successor to the closest successor in the other set of successors, here: the distance from y_3 to x_3).

In order to model such transition systems as coalgebras we define the following n-ary auxiliary functor: $G(X_1,\ldots,X_n) = \{u\colon \Sigma \to X_1 + \cdots + X_n \mid u(r_i) \in X_i\}$. Then coalgebras

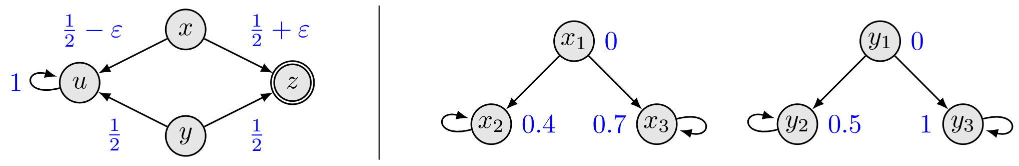


Figure 1 A probabilistic transition system (left) and a metric transition system (right).

are of the form $\alpha\colon S \to G(M_{r_1},\dots,M_{r_n}) \times \mathcal{P}_{\mathit{fin}}(S)$, where $\mathcal{P}_{\mathit{fin}}$ is the finite powerset functor and $\alpha(s) = ([s], \tau(s))$. As we will see later in Example 6.7, the right-hand side of (1) can be seen as lifting a metric d on X to a metric on $G(M_{r_1},\dots,M_{r_n}) \times \mathcal{P}_{\mathit{fin}}(X)$.

Generalizing from the examples, we now establish a general framework for deriving such behavioral distances. In both cases, the crucial step is to find, for a functor F, a way to lift a pseudometric on X to a pseudometric on FX. Based on this, one can set up a fixed-point equation and define behavioral distance as its smallest solution. Hence, in the next sections we describe how to lift an endofunctor F on **Set** to an endofunctor on **PMet**.

▶ **Definition 2.7** (Lifting)**.** Let $U\colon \mathbf{PMet} \to \mathbf{Set}$ be the forgetful functor which maps every pseudometric space to its underlying set. A functor $\overline{F}\colon \mathbf{PMet} \to \mathbf{PMet}$ is called a *lifting* of a functor $F\colon \mathbf{Set} \to \mathbf{Set}$ if it satisfies $U \circ \overline{F} = F \circ U$.

It is not difficult to prove that such a lifting is always monotone on pseudometrics over a common set, i.e. for any two pseudometrics $d_1 \le d_2$ on the same set X, we also have $d_1^F \le d_2^F$ where d_i^F are the pseudometrics on FX obtained by applying $\overline{F}$ to (X, d_i) (see [3]). Similarly to predicate lifting of coalgebraic modal logic [18], liftings on **PMet** can be conveniently defined via an evaluation function.

▶ **Definition 2.8** (Evaluation Function and Evaluation Functor)**.** Let F be an endofunctor on **Set**. An *evaluation function* for F is a function $ev_F\colon F[0,\top] \to [0,\top]$. Given such a function, we define the *evaluation functor* to be the endofunctor $\widetilde{F}$ on $\mathbf{Set}/[0,\top]$, the slice category[1] over $[0,\top]$, via $\widetilde{F}(g) = ev_F \circ Fg$ for all $g \in \mathbf{Set}/[0,\top]$. On arrows $\widetilde{F}$ is defined as F.

3 Lifting Functors to Pseudometric Spaces à la Kantorovich

Let us now consider an endofunctor F on **Set** with an evaluation function ev_F. Given a pseudometric space (X,d), our first approach will be to take the smallest possible pseudometric d^F on FX such that, for all nonexpansive functions $f\colon (X,d) \xrightarrow{1} ([0,\top], d_e)$, also $\widetilde{F}f\colon (FX, d^F) \xrightarrow{1} ([0,\top], d_e)$ is nonexpansive again, i.e. we want to ensure that for all $t_1, t_2 \in FX$ we have $d_e(\widetilde{F}f(t_1), \widetilde{F}f(t_2)) \le d^F(t_1,t_2)$. This idea immediately leads us to the next definition.

▶ **Definition 3.1** (Kantorovich Pseudometric and Kantorovich Lifting)**.** Let $F\colon \mathbf{Set} \to \mathbf{Set}$ be a functor with an evaluation function ev_F. For every pseudometric space (X,d) the *Kantorovich pseudometric* on FX is the function $d^{\uparrow F}\colon FX \times FX \to [0,\top]$, where for all $t_1, t_2 \in FX$:

$$d^{\uparrow F}(t_1,t_2) := \sup\{d_e(\widetilde{F}f(t_1), \widetilde{F}f(t_2)) \mid f\colon (X,d) \xrightarrow{1} ([0,\top], d_e)\}\,.$$

[1] The slice category $\mathbf{Set}/[0,\top]$ has as objects all functions $g\colon X \to [0,\top]$ where X is an arbitrary set. Given g as before and $h\colon Y \to [0,\top]$, an arrow from g to h is a function $f\colon X \to Y$ satisfying $h \circ f = g$.

The *Kantorovich lifting* of the functor F is the functor $\overline{F}\colon \mathbf{PMet} \to \mathbf{PMet}$ defined as $\overline{F}(X,d) = (FX, d^{\uparrow F})$ and $\overline{F}f = Ff$.

It is easy to show that $d^{\uparrow F}$ is indeed a pseudometric. Since $\overline{F}$ inherits the preservation of identities and composition of morphisms from F we can prove that nonexpansive functions are mapped to nonexpansive functions and isometries to isometries.

▶ **Proposition 3.2.** *The Kantorovich lifting $\overline{F}$ of a functor F preserves isometries.*

We chose the name *Kantorovich* because our definition is reminiscent of the Kantorovich pseudometric in probability theory. If we take the proper combination of functor and evaluation function, we can recover that pseudometric (in the discrete case) as the first instance for our framework.

▶ **Example 3.3** (Probability Distribution Functor)**.** We take $\top = 1$, the probability distribution functor $\mathcal{D}$ from Example 2.5 and define $ev_{\mathcal{D}}\colon \mathcal{D}[0,1] \to [0,1]$, $ev_{\mathcal{D}}(P) = \mathbb{E}_P[\mathrm{id}_{[0,1]}] = \sum_{x\in[0,1]} x \cdot P(x)$ yielding $\widetilde{\mathcal{D}}g(P) = \mathbb{E}_P[g] = \sum_{x\in[0,1]} g(x) \cdot P(x)$ for all $g\colon X \to [0,1]$. For every pseudometric space (X,d) we obtain the Kantorovich pseudometric $d^{\uparrow\mathcal{D}}\colon (\mathcal{D}X)^2 \to [0,1]$, $d^{\uparrow\mathcal{D}}(P_1,P_2) = \sup\{\sum_{x\in X} f(x) \cdot \big(P_1(x) - P_2(x)\big) \mid f\colon (X,d) \xrightarrow{1} ([0,1], d_e)\}$.

In general Kantorovich liftings do not preserve metrics, as shown by the following example.

▶ **Example 3.4.** Let $F\colon \mathbf{Set} \to \mathbf{Set}$ be given as $FX = X \times X$ on sets and $Ff = f \times f$ on functions and take $\top = \infty$, $ev_F\colon F[0,\infty] \to [0,\infty], ev_F(r_1,r_2) = r_1 + r_2$. For a metric space (X,d) with $|X| \geq 2$ let $t_1 = (x_1,x_2) \in FX$ with $x_1 \neq x_2$ and define $t_2 := (x_2,x_1)$. Clearly $t_1 \neq t_2$ but for every nonexpansive function $f\colon (X,d) \xrightarrow{1} ([0,\top], d_e)$ we have $\widetilde{F}f(t_1) = f(x_1) + f(x_2) = f(x_2) + f(x_1) = \widetilde{F}f(t_2)$ and thus $d^{\uparrow F}(t_1,t_2) = 0$.

4 Wasserstein Pseudometric and Kantorovich-Rubinstein Duality

We have seen that our first lifting approach bears close resemblance to the original Kantorovich pseudometric on probability measures. In that context there exists another pseudometric, the Wasserstein pseudometric, which under certain conditions coincides with the Kantorovich pseudometric. We will define a generalized version of the Wasserstein pseudometric and compare it with our generalized Kantorovich pseudometric. To do that we first need to define how we can couple elements of FX.

▶ **Definition 4.1** (Coupling)**.** Let $F\colon \mathbf{Set} \to \mathbf{Set}$ be a functor and $n \in \mathbb{N}$. Given a set X and $t_i \in FX$ for $1 \leq i \leq n$ we call an element $t \in F(X^n)$ such that $F\pi_i(t) = t_i$ a *coupling* of the t_i (with respect to F). We write $\Gamma_F(t_1, t_2, \dots, t_n)$ for the set of all these couplings.

If F preserves weak pullbacks, we can define new couplings based on given ones.

▶ **Lemma 4.2** (Gluing Lemma)**.** *Let $F\colon \mathbf{Set} \to \mathbf{Set}$ be a weak pullback preserving functor, X a set, $t_1, t_2, t_3 \in FX$, $t_{12} \in \Gamma_F(t_1,t_2)$, and $t_{23} \in \Gamma_F(t_2,t_3)$ be couplings. Then there is a coupling $t_{123} \in \Gamma_F(t_1,t_2,t_3)$ such that $F(\langle\pi_1^3, \pi_2^3\rangle)(t_{123}) = t_{12}$ and $F(\langle\pi_2^3, \pi_3^3\rangle)(t_{123}) = t_{23}$.*

This lemma already hints at the fact that our new lifting will only work for weak pullback preserving functors, which is a standard requirement in coalgebra. In addition to that we have to impose three extra conditions on the evaluation functions.

▶ **Definition 4.3** (Well-Behaved Evaluation Function)**.** Let ev_F be an evaluation function for a functor $F\colon \mathbf{Set} \to \mathbf{Set}$. We call ev_F *well-behaved* if it satisfies the following conditions:

1. $\widetilde{F}$ is monotone, i.e., for $f,g\colon X\to[0,\top]$ with $f\le g$, we have $\widetilde{F}f\le\widetilde{F}g$.
2. For each $t\in F([0,\top]^2)$ it holds that $d_e(ev_F(t_1),ev_F(t_2))\le\widetilde{F}d_e(t)$ for $t_i:=F\pi_i(t)$.
3. $ev_F^{-1}[\{0\}]=Fi[F\{0\}]$ where $i\colon\ \{0\}\hookrightarrow[0,\top]$ is the inclusion map.

While the first condition of this definition is quite natural, the other two need to be explained. Condition 2 is needed to ensure that $\widetilde{F}\mathrm{id}_{[0,\top]}=ev_F\colon F[0,\top]\to[0,\top]$ is nonexpansive once d_e is lifted to $F[0,\top]$ (cf. the intuition behind the Kantorovich lifting, where we ensure that $\widetilde{F}f$ is nonexpansive whenever f is nonexpansive). Furthermore Condition 3 intuitively says that exactly the elements of $F\{0\}$ are mapped to 0 via ev_F. Before we define the Wasserstein pseudometric and the corresponding lifting, we take a look at an example of a functor together with a well-behaved evaluation function.

▶ **Example 4.4** (Finite Powerset Functor). Let $\top=\infty$. We take the finite powerset functor $\mathcal{P}_{\mathit{fin}}$ with evaluation function $\max\colon\mathcal{P}_{\mathit{fin}}([0,\infty])\to[0,\infty]$ with $\max\emptyset=0$. This evaluation function is well-behaved whereas $\min\colon\mathcal{P}_{\mathit{fin}}([0,\infty])\to[0,\infty]$ is not well-behaved.

▶ **Definition 4.5** (Wasserstein Pseudometric and Wasserstein Lifting). Let $F\colon\mathbf{Set}\to\mathbf{Set}$ be a weak pullback preserving functor with well-behaved evaluation function ev_F. For every pseudometric space (X,d) the *Wasserstein pseudometric* on FX is the function $d^{\downarrow F}\colon FX\times FX\to[0,\top]$ given by, for all $t_1,t_2\in FX$,

$$d^{\downarrow F}(t_1,t_2):=\inf\{\widetilde{F}d(t)\mid t\in\Gamma_F(t_1,t_2)\}\,.$$

We define the *Wasserstein lifting* of F to be the functor $\overline{F}\colon\mathbf{PMet}\to\mathbf{PMet}$, $\overline{F}(X,d)=(FX,d^{\downarrow F})$, $\overline{F}f=Ff$.

This time it is not straightforward to prove that $d^{\downarrow F}$ is a pseudometric, so we explicitly provide the following result. Its proof relies on all properties of well-behavedness of ev_F and uses Lemma 4.2 which explains why we need a weak pullback preserving functor.

▶ **Proposition 4.6.** *The Wasserstein pseudometric is a well-defined pseudometric on FX.*

It is not hard to show functoriality of $\overline{F}$ and, as before, the lifted functor preserves isometries.

▶ **Proposition 4.7.** *The Wasserstein lifting $\overline{F}$ of a functor F preserves isometries.*

In contrast to our previous approach, metrics are preserved in certain situations.

▶ **Proposition 4.8** (Preservation of Metrics). *Let (X,d) be a metric space and F be a functor. If the infimum in Definition 4.5 is a minimum for all $t_1,t_2\in FX$ where $d^{\downarrow F}(t_1,t_2)=0$ then $d^{\downarrow F}$ is a metric, thus also $\overline{F}(X,d)=(FX,d^{\downarrow F})$ is a metric space.*

Please note that a similar restriction for the Kantorovich lifting (i.e. requiring that the supremum in Definition 3.1 is a maximum) does not yield preservation of metrics: In Example 3.4 the supremum is always a maximum but we do not get a metric.

Let us now compare both lifting approaches. Whenever it is defined, the Wasserstein pseudometric is an upper bound for the Kantorovich pseudometric.

▶ **Proposition 4.9.** *Let F be a weak pullback preserving functor with well-behaved evaluation function. Then for all pseudometric spaces (X,d) it holds that $d^{\uparrow F}\le d^{\downarrow F}$.*

In general this inequality may be strict in general, as the following example shows.

▶ **Example 4.10.** The functor of Example 3.4 preserves weak pullbacks and the evaluation function is well-behaved. We continue the example and take $t_1 = (x_1, x_2)$, $t_2 = (x_2, x_1)$. The unique coupling $t \in \Gamma_F(t_1, t_2)$ is $t = (x_1, x_2, x_2, x_1)$. Using that d is a metric we conclude that $d^{\downarrow F}(t_1, t_2) = \widetilde{F}d(t) = d(x_1, x_2) + d(x_2, x_1) = 2d(x_1, x_2) > 0 = d^{\uparrow F}(t_1, t_2)$.

When the inequality can be replaced by an equality we will in the following say that the Kantorovich-Rubinstein duality holds. In this case we obtain a canonical notion of distance on FX, given a pseudometric space (X, d). To calculate the distance of $t_1, t_2 \in FX$ it is then enough to find a nonexpansive function $f\colon (X, d) \xrightarrow{1} ([0, \top], d_e)$ and a coupling $t \in \Gamma_F(t_1, t_2)$ such that $d_e(\widetilde{F}f(t_1), \widetilde{F}f(t_2)) = \widetilde{F}d_e(t)$. Then, due to Proposition 4.9, this value equals $d^{\uparrow F}(t_1, t_2) = d^{\downarrow F}(t_1, t_2)$. We will now take a look at some examples where the duality holds.

▶ **Example 4.11** (Identity Functor)**.** Take $F = \mathrm{Id}$ with the identity evaluation map $ev_{\mathrm{Id}} = \mathrm{id}_{[0,\top]}$. For any $t_1, t_2 \in X$, $t := (t_1, t_2)$ is the unique coupling of t_1, t_2. Hence, $d^{\downarrow F}(t_1, t_2) = d(t_1, t_2)$. With the function $d(t_1, \_)\colon (X, d) \xrightarrow{1} ([0, \top], d_e)$ we obtain duality because we have $d(t_1, t_2) = d_e(d(t_1, t_1), d(t_1, t_2)) \leq d^{\uparrow F}(t_1, t_2) \leq d^{\downarrow F}(t_1, t_2) = d(t_1, t_2)$ and thus equality. Similarly, if we define $ev_{\mathrm{Id}}(r) = c \cdot r$ for $r \in [0, \top]$, $0 < c \leq 1$, the Kantorovich and Wasserstein liftings coincide and we obtain the discounted distance $d^{\uparrow F}(t_1, t_2) = d^{\downarrow F}(t_1, t_2) = c \cdot d(t_1, t_2)$.

▶ **Example 4.12** (Probability Distribution Functor)**.** The functor $\mathcal{D}$ of Example 3.3 preserves weak pullbacks [19] and the evaluation function $ev_{\mathcal{D}}$ is well-behaved. We recover the usual Wasserstein pseudometric $d^{\downarrow \mathcal{D}}(P_1, P_2) = \inf\{\sum_{x_1, x_2 \in X} d(x_1, x_2) \cdot P(x_1, x_2) \mid P \in \Gamma_{\mathcal{D}}(P_1, P_2)\}$ and the Kantorovich-Rubinstein duality [24] from transportation theory for the discrete case.

▶ **Example 4.13** (Finite Powerset Functor and Hausdorff Pseudometric)**.** Let $\top = \infty$, $F = \mathcal{P}_{\mathit{fin}}$ with evaluation map $ev_{\mathcal{P}_{\mathit{fin}}}\colon \mathcal{P}_{\mathit{fin}}([0, \infty]) \to [0, \infty]$, $ev_{\mathcal{P}_{\mathit{fin}}}(R) = \max R$ with $\max \emptyset = 0$ (as in Example 4.4). In this setting we obtain duality and both pseudometrics are equal to the Hausdorff pseudometric d_H on $\mathcal{P}_{\mathit{fin}}(X)$ which is defined as, for all $X_1, X_2 \in \mathcal{P}_{\mathit{fin}}(X)$,

$$d_H(X_1, X_2) = \max\left\{ \max_{x_1 \in X_1} \min_{x_2 \in X_2} d(x_1, x_2), \max_{x_2 \in X_2} \min_{x_1 \in X_1} d(x_1, x_2) \right\}.$$

Note that the distance is ∞, if either X_1 or X_2 is empty.

It is also illustrative to consider the countable powerset functor. Using the supremum as evaluation function, one obtains again the Hausdorff pseudometric (with supremum/infimum replacing maximum/minimum). However, in this case the Hausdorff distance of different countable sets might be 0, even if we lift a metric. This shows that in general the Wasserstein lifting does not preserve metrics but we need an extra condition, e.g. the one in Proposition 4.8.

5 Lifting Multifunctors

Our two approaches can easily be generalized[2] to lift a multifunctor $F\colon \mathbf{Set}^n \to \mathbf{Set}$ (for $n \in \mathbb{N}$) in a similar sense as given by Definition 2.7 to a multifunctor $\overline{F}\colon \mathbf{PMet}^n \to \mathbf{Set}$. The only difference is that we start with n pseudometric spaces instead of one. Now we need an *evaluation function* $ev_F\colon F([0, \top], \ldots, [0, \top]) \to [0, \top]$ which we call *well-behaved* if it satisfies conditions similar to Definition 4.3 and which gives rise to an evaluation multifunctor $\widetilde{F}\colon (\mathbf{Set}/[0, \top])^n \to \mathbf{Set}/[0, \top]$. Given $t_1, t_2 \in F(X_1, \ldots, X_n)$ we

[2] The details are spelled out in [3], here we provide just the basic ideas.

write again $\Gamma_F(t_1,t_2) \subseteq F(X_1^2,\dots,X_n^2)$ for the set of couplings which is defined analogously to Definition 4.1. For pseudometrics $d_i\colon X_i^2 \to [0,\top]$, we can then define the *Kantorovich/Wasserstein pseudometric* $d^{\uparrow F}_{1,\dots,n}, d^{\downarrow F}_{1,\dots,n}\colon F(X_1,\dots,X_n)\times F(X_1,\dots,X_n) \to [0,\top]$, as $d^{\uparrow F}_{1,\dots,n}(t_1,t_2) := \sup\{d_e(\widetilde{F}(f_1,\dots,f_n)(t_1), \widetilde{F}(f_1,\dots,f_n)(t_2)) \mid f_i\colon (X_i,d_i) \xrightarrow{1} ([0,\top],d_e)\}$ and $d^{\downarrow F}_{1,\dots,n}(t_1,t_2) := \inf\{\widetilde{F}(d_1,\dots,d_n)(t) \mid t \in \Gamma_F(t_1,t_2)\}$. This setting grants us access to new examples such as the product and the coproduct bifunctors.

▶ **Example 5.1** (Product Bifunctor)**.** For the product bifunctor $F\colon \mathbf{Set}^2 \to \mathbf{Set}$ where $F(X_1,X_2) = X_1 \times X_2$ and $F(f_1,f_2) = f_1 \times f_2$ we consider the evaluation function $\max\colon [0,\top]^2 \to [0,\top]$ and for fixed parameters $c_1,c_2 \in (0,1]$ and $p \in \mathbb{N}$ the function $\rho\colon [0,\top]^2 \to [0,\top]$, $\rho(x_1,x_2) = (c_1x_1^p + c_2x_2^p)^{1/p}$. These functions are well-behaved, the Kantorovich-Rubinstein duality holds and the supremum [infimum] of the Kantorovich [Wasserstein] pseudometrics is always a maximum [minimum]. For the first function we obtain the ∞-product pseudometric $d_\infty((x_1,x_2),(y_1,y_2)) = \max(d_1(x_1,y_1), d_2(x_2,y_2))$ and for the other function the weighted p-product pseudometric $d_p((x_1,x_2),(y_1,y_2)) = (c_1d_1^p(x_1,y_1) + c_2d_2^p(x_2,y_2))^{1/p}$.

Note that the pseudometric space $(X_1 \times X_2, d_\infty)$ is the usual binary (category theoretic) product of (X_1,d_1) and (X_2,d_2). Similarly, we can also obtain the binary coproduct.

▶ **Example 5.2** (Coproduct Bifunctor)**.** For the coproduct bifunctor $F\colon \mathbf{Set}^2 \to \mathbf{Set}$, where $F(X_1,X_2) = X_1 + X_2 = X_1 \times \{1\} \cup X_2 \times \{2\}$ and $F(f_1,f_2) = f_1 + f_2$ we take the evaluation function $ev_F\colon [0,\top] + [0,\top] \to [0,\top]$, $ev_F(x,i) = x$. This function is well-behaved, the Kantorovich-Rubinstein duality holds and the supremum of the Kantorovich pseudometric is always a maximum whereas the infimum of the Wasserstein pseudometric is a minimum if and only if any coupling of the two elements exists. We obtain the coproduct pseudometric d_+ where $d_+((x_1,i_1),(x_2,i_2))$ is equal to $d_i(x_1,x_2)$ if $i_1 = i_2 = i$ and equal to $\top$ otherwise.

6 Final Coalgebra and Coalgebraic Behavioral Pseudometrics

In this section we assume an arbitrary lifting $\overline{F}\colon \mathbf{PMet} \to \mathbf{PMet}$ of an endofunctor F on $\mathbf{Set}$. For any pseudometric space (X,d) we write d^F for the pseudometric obtained by applying $\overline{F}$ to (X,d). Such a lifting can be obtained as described earlier, but also by taking a lifted multifunctor and fixing all parameters apart from one, or by the composition of such functors. The following result ensures that if $\kappa\colon \Omega \to F\Omega$ is a final F-coalgebra, then there is also a final $\overline{F}$-coalgebra which is constructed by simply enriching Ω with a pseudometric d_Ω.

▶ **Theorem 6.1.** *Let $\overline{F}\colon \mathbf{PMet} \to \mathbf{PMet}$ be a lifting of a functor $F\colon \mathbf{Set} \to \mathbf{Set}$ which has a final coalgebra $\kappa\colon \Omega \to F\Omega$. For every ordinal i we construct a pseudometric $d_i\colon \Omega\times\Omega \to [0,\top]$ as follows: $d_0 := 0$ is the zero pseudometric, $d_{i+1} := d_i^F \circ (\kappa \times \kappa)$ for all ordinals i and $d_j = \sup_{i<j} d_i$ for all limit ordinals j. This sequence converges for some ordinal θ, i.e $d_\theta = d_\theta^F \circ (\kappa \times \kappa)$. Moreover $\kappa\colon (\Omega, d_\theta) \xrightarrow{1} (F\Omega, d_\theta^F)$ is the final $\overline{F}$-coalgebra.*

We noted that for any set X, the set of pseudometrics over X, with pointwise order, is a complete lattice. Moreover the lifting $\overline{F}$ induces a monotone function $\_^F$ which maps any pseudometric d on X to d^F on FX. If, additionally, such function is ω-continuous, i.e., it preserves the supremum of ω-chains, the construction in Theorem 6.1 will converge in at most ω steps, i.e., $d_\theta = d_\omega$. We show in [3] that the liftings induced by the finite powerset functor and the probability distribution functor with finite support are ω-continuous. The arguments used for convergence here suggests a connection with the work in [20], which provides fixed-point results for metric functors which are not locally contractive.

Beyond equivalences of states, in **PMet** we can measure the distance of behaviors in the final coalgebra. More precisely, the *behavioral distance* of two states $x, y \in X$ of some coalgebra $\alpha\colon X \to FX$ is defined via the pseudometric $bd(x,y) = d_\theta([\![x]\!], [\![y]\!])$. Such distances can be computed analogously to d_θ above, replacing $\kappa\colon \Omega \to F\Omega$ by α. This way we do not need to explore the entire final coalgebra (which might be too large) but can restrict to the interesting part.

▶ **Theorem 6.2.** *Let the chain of the d_i converge in θ steps and $\overline{F}$ preserve isometries. Let furthermore $\alpha\colon X \to FX$ be an arbitrary coalgebra. For all ordinals i we define a pseudometric $e_i\colon X \times X \to [0,\top]$ as follows: e_0 is the zero pseudometric, $e_{i+1} = e_i^F \circ (\alpha \times \alpha)$ for all ordinals i and $e_j = \sup_{i<j} e_i$ for all limit ordinals j. Then we reach a fixed point after $\zeta \leq \theta$ steps, i.e. $e_\zeta = e_\zeta^F \circ (\alpha \times \alpha)$, such that $bd = e_\zeta$.*

Since d_θ is a pseudometric, we have that if $[\![x]\!] = [\![y]\!]$ then $bd(x,y) = 0$. The other direction does not hold in general: for this d_θ has to be a proper metric. Theorem 6.5 at the end of this section provides sufficient conditions guaranteeing this property.

To this aim, we proceed by recalling the final coalgebra construction via the final chain which was first presented in the dual setting (free/initial algebra).

▶ **Definition 6.3** (Final Coalgebra Construction [1]). Let **C** be a category with terminal object **1** and limits of ordinal-indexed cochains. For any functor $F\colon \mathbf{C} \to \mathbf{C}$ the *final chain* consists of objects W_i for all ordinals i and *connection morphisms* $p_{i,j}\colon W_j \to W_i$ for all ordinals $i \leq j$. The objects are defined as $W_0 := \mathbf{1}$, $W_{i+1} := FW_i$ for all ordinals i, and $W_j := \lim_{i<j} W_i$ for all limit ordinals j. The morphisms are determined by $p_{0,i} := !\colon W_i \to \mathbf{1}$, $p_{i,i} = \mathrm{id}_{W_i}$ for all ordinals i, $p_{i+1,j+1} := Fp_{i,j}$ for all ordinals $i < j$ and if j is a limit ordinal the $p_{i,j}$ are the morphisms of the limit cone. They satisfy $p_{i,k} = p_{i,j} \circ p_{j,k}$ for all ordinals $i \leq j \leq k$. We say that the chain *converges* in λ steps if $p_{\lambda,\lambda+1}\colon W_{\lambda+1} \to W_\lambda$ is an iso.

This construction does not necessarily converge, but if it does, we get a final coalgebra.

▶ **Proposition 6.4** ([1]). *Let **C** be a category with terminal object **1** and limits of ordinal-indexed cochains. If the final chain of a functor $F\colon \mathbf{C} \to \mathbf{C}$ converges in λ steps then $p^{-1}_{\lambda,\lambda+1}\colon W_\lambda \to FW_\lambda$ is the final coalgebra.*

We now show under which circumstances d_θ is a metric and how our construction relates to the construction of the final chain.

▶ **Theorem 6.5.** *Let $\overline{F}\colon \mathbf{PMet} \to \mathbf{PMet}$ be a lifting of a functor $F\colon \mathbf{Set} \to \mathbf{Set}$ which has a final coalgebra $\kappa\colon \Omega \to F\Omega$. Assume that $\overline{F}$ preserves isometries and metrics, that the final chain for F converges and the chain of the d_i converges in θ steps. Then d_θ is a metric, i.e. for $x, y \in \Omega$ we have $d_\theta(x,y) = 0 \iff x = y$.*

We will now get back to the examples studied at the beginning of the paper (Example 2.5 and Example 2.6) and discuss in which sense they are instances of our framework.

▶ **Example 6.6** (Probabilistic Transition System, revisited). To model the behavioral distance from Example 2.5 in our framework, we set $\top = 1$ and proceed to lift the following three functors: we first consider the identity functor Id with evaluation map $ev_{\mathrm{Id}}\colon [0,1] \to [0,1]$, $ev_{\mathrm{Id}}(z) = c \cdot z$ in order to integrate the discount (Example 4.11). Then, we take the coproduct with the singleton metric space (Example 5.2). The combination of the two functors yields the discrete version of the refusal functor of [23], namely $\overline{R}(X,d) = (X + \mathbf{1}, \hat{d})$ where $\hat{d}$ is taken from Example 2.5. Finally, we lift the probability distribution functor $\mathcal{D}$ to obtain $\overline{\mathcal{D}}$ (Example 3.3). All functors satisfy the Kantorovich-Rubinstein duality and preserve metrics.

It is readily seen that $\overline{\mathcal{D}}(\overline{R}(X,d)) = (\mathcal{D}(X+1), \overline{d})$, where $\overline{d}$ is defined as in Example 2.5). Then, the least solution of $d(x,y) = \overline{d}(\alpha(x), \alpha(y))$ can be computed as in Theorem 6.2.

▶ **Example 6.7** (Metric Transition Systems, revisited)**.** To obtain propositional distances in metric transition systems we set $\top = \infty$. We also define, for the auxiliary functor G, an evaluation function $ev_G\colon G([0,\infty],\dots,[0,\infty]) \to [0,\infty]$ with $ev_G(u) = \max_{r\in\Sigma} u(r)$. Let $\overline{G}$ be the corresponding lifted functor. It can be shown, similarly to Example 5.1, that the Kantorovich-Rubinstein duality holds and metrics are preserved. We instantiate the given pseudometric spaces (M_{r_i}, d_{r_i}) as parameters and obtain the functor $\overline{F}(X,d) = \overline{G}((M_{r_1}, d_{r_1}), \dots, (M_{r_n}, d_{r_n})) \times \overline{\mathcal{P}_{\mathit{fin}}}(X,d)$ (for the lifting of the powerset functor see Example 4.13). Then, via Theorem 6.2, we obtain exactly the least solution of (1) in Example 2.6.

7 Related and Future Work

The ideas for our framework are heavily influenced by work on quantitative variants of (bisimulation) equivalence of probabilistic systems. In that context at first Giacalone et al. [12] observed that probabilistic bisimulation [16] is too strong and therefore introduced a metric based on the notion of ε-bisimulations.

Using a logical characterization of bisimulation for labelled Markov processes (LMP) [8], Desharnais et al. defined a family of metrics between these LMPs [9] via functional expressions: if evaluated on a state of an LMP, such a functional expression measures the extent to which a formula is satisfied in that state. A different, coalgebraic approach, which inspired ours, is used by van Breugel et al. [23]. As presented in more detail in the examples above, they define a pseudometric on probabilistic systems via the Kantorovich pseudometric for probability measures. Moreover, they show in [22] that this metric is related to the logical pseudometric by Desharnais et al.

Our framework provides a toolbox to determine behavioral distances for different types of transition systems modeled as coalgebras. Moreover, the liftings introduced in this paper pave the way to extend several coalgebraic methods to reason about quantitative properties of systems. For instance the bisimulation proof principle, which allows to check behavioral equivalence, assumes a specific meaning in **PMet**: every coalgebra $\alpha\colon (X,d) \to \overline{F}(X,d)$ *coinductively proves* that the behavioral distance bd of the underlying F-coalgebra on **Set** is smaller or equal than d. Indeed, since $[\![\_]\!]$ is nonexpansive, $d \geq d_\theta([\![\_]\!], [\![\_]\!]) = bd$. This principle, which has already been stated in different formulations (see e.g. [7, 10, 21]), can now be enhanced via *up-to techniques* by exploiting the liftings introduced in this paper and the coalgebraic understanding of such enhancements given in [4].

Since up-to techniques can exponentially improve algorithms for equivalence-checking, we hope that they could also optimize some of the algorithms for computing (or approximating) behavioral distances [23, 21, 5, 2]. At this point, it is worth recalling that the Kantorovich-Rubinstein duality has been exploited in [23] for defining one of these algorithms: the characterization given by the Wasserstein metric allows to reduce to linear programming.

Another line of research potentially stemming from our work concerns the so-called *abstract GSOS* [15] which provides abstract coalgebraic conditions ensuring compositionality of behavioral equivalence (with respect to some operators). By taking our lifting to **PMet**, abstract GSOS guarantees the nonexpansiveness of behavioral distance, a property that has captured the interest of several researchers [9, 11]. The main technical challenge would be to lift to **PMet** not only functors, but also distributive laws. Lifting of distributive laws would also be needed for defining *linear behavioral distances*, exploiting the coalgebraic account of trace semantics based on Kleisli categories [14].

We finally observe that the chains of Theorems 6.1 and 6.2 can be understood in terms of fibrations along the lines of [13]. A detailed comparison with [13] can be found in [3].

Acknowledgements. The authors are grateful to Franck van Breugel, Neil Ghani and Daniela Petrişan for several precious suggestions and inspiring discussions. The second author acknowledges the support by project `ANR 12IS0 2001 PACE`.

References

1 Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015(4):589–602, 1974.

2 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. In *Proc. of TACAS'13*, pages 1–15, 2013.

3 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Behavioral metrics via functor lifting. Extended version with proofs, see `arXiv:1410.3385`, 2014.

4 Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Coinduction up to in a fibrational setting. In *Proc. of CSL-LICS'14*, 2014.

5 Di Chen, Franck van Breugel, and James Worrell. On the complexity of computing probabilistic bisimilarity. In *Proc. of FoSSaCS'12*, volume 7213 of *LNCS*, pages 437–451, 2012.

6 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Transactions on Software Engineering*, 25(2), 2009.

7 Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. *ENTCS*, 153(2):79–96, 2006.

8 Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179(2):163–193, 2002.

9 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *TCS*, 318(3):323–354, 2004.

10 Joseé Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proc. of LICS'02*, pages 413–422. IEEE, 2002.

11 Daniel Gebler and Simone Tini. Compositionality of approximate bisimulation for probabilistic systems. In *Proc. of EXPRESS/SOS'13*, pages 32–46, 2013.

12 Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proc. IFIP TC2 Working Conference on Programming Concepts and Methods*, pages 443–458. North-Holland, 1990.

13 Ichiro Hasuo, Kenta Cho, Toshiki Kataoka, and Bart Jacobs. Coinductive predicates and final sequences in a fibration. *ENTCS*, 298:197–214, 2013.

14 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *LMCS*, 3 (4:11):1–36, November 2007.

15 Bartek Klin. Bialgebras for structural operational semantics: An introduction. *TCS*, 412(38):5043–5069, 2011.

16 Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. In *Proc. of POPL'89*, pages 344–352, 1989.

17 J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *TCS*, 249:3–80, 2000.

18 Lutz Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *TCS*, 390(2–3):230–247, 2008.

19 Ana Sokolova. Probabilistic systems coalgebraically: A survey. *TCS*, 412(38):5095–5110, 2011. CMCS Tenth Anniversary Meeting.

20 Franck van Breugel, Claudio Hermida, Michael Makkai, and James Worrell. Recursively defined metric spaces without contraction. *TCS*, 380(1–2):143–163, July 2007.

21 Franck van Breugel, Babita Sharma, and James Worrell. Approximating a behavioural pseudometric without discount for probabilistic systems. *LMCS*, 4(2), 2008.

22 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *TCS*, 331:115–142, 2005.

23 Franck van Breugel and James Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *TCS*, 360(1):373–385, 2006.

24 Cédric Villani. *Optimal Transport – Old and New*, volume 338 of *A Series of Comprehensive Studies in Mathematics*. Springer, 2009.

Foundation of Diagnosis and Predictability in Probabilistic Systems*

Nathalie Bertrand[1], **Serge Haddad**[2], **and Engel Lefaucheux**[1,2]

1 Inria, France `nathalie.bertrand@inria.fr`
2 LSV, ENS Cachan & CNRS & Inria, France
`{serge.haddad,engel.lefaucheux}@ens-cachan.fr`

Abstract

In discrete event systems prone to unobservable faults, a diagnoser must eventually detect fault occurrences. The diagnosability problem consists in deciding whether such a diagnoser exists. Here we investigate diagnosis for probabilistic systems modelled by partially observed Markov chains also called probabilistic labeled transition systems (pLTS). First we study different specifications of diagnosability and establish their relations both in finite and infinite pLTS. Then we analyze the complexity of the diagnosability problem for finite pLTS: we show that the polynomial time procedure earlier proposed is erroneous and that in fact for all considered specifications, the problem is PSPACE-complete. We also establish tight bounds for the size of diagnosers. Afterwards we consider the dual notion of predictability which consists in predicting that in a safe run, a fault will eventually occur. Predictability is an easier problem than diagnosability: it is NLOGSPACE-complete. Yet the predictor synthesis is as hard as the diagnoser synthesis. Finally we introduce and study the more flexible notion of *prediagnosability* that generalizes predictability and diagnosability.

1998 ACM Subject Classification D.2.5 Testing and Debugging

Keywords and phrases Partially observed systems, Diagnosis, Markov chains

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.417

1 Introduction

Diagnosis. In computer science, diagnosis may refer to different kinds of activities. For instance, in artificial intelligence it can describe the process of identifying a disease from its symptoms, as performed by the expert system MYCIN [3]. In this work, we concentrate on diagnosis as studied in control theory, where it is applied to partially observable systems prone to faults. A sequence of observations of such a system is said to be surely correct (respectively surely faulty) if all possible runs corresponding to this sequence are correct (respectively faulty); otherwise the observed sequence is ambiguous. While monitoring the system, the *diagnoser* should rule out ambiguities, and in particular detect that a fault occurred; and the problem of existence of such a diagnoser is refered to as *diagnosability* [12]. In order to anticipate problems triggered by fault occurrences, one can also be interested in *predictors* that detect that a fault will eventually occur, and the *predictability* problem [6] is concerned with the existence of a predictor.

* This work has been supported by project ImpRo ANR-2010-BLAN-0317 and the European Union Seventh Framework Programme [FP7/2007-2013] under grant agreement 257462 HYCON2 NOE.

Diagnosis of discrete event systems. Diagnosability and predictability were first defined and studied in the framework of finite discrete event systems modelled by labeled transition systems (LTS), and the problems were shown to be solvable in PTIME (see [8] and [6], respectively). Despite the polynomial time complexity of the decision problems, for diagnosable (respectively predictable) LTS, the size of the diagnoser (respectively predictor) constructed by the algorithms may be exponential. Diagnosers as well as predictors must ensure two requirements: *correctness*, meaning that the information provided by the diagnoser/predictor is accurate, and *reactivity*, ensuring that a fault will eventually be detected.

Diagnosis of probabilistic systems. Building on the work for LTS, the notion of diagnosability was later extended to Markov chains with labels on transitions, also called probabilitic labeled transition systems (pLTS) [13]. In a probabilistic context, the reactivity requirement now asks that faults will be almost surely eventually detected. Regarding correctness, two specifications have been proposed: either one sticks to the original definition and requires that the provided information is accurate, defining *A-diagnosability*; or one weakens the correctness by admitting errors in the provided information that should, however, have an arbitrary small probability when the delay before the diagnostic is long enough, defining *AA-diagnosability.* From a computational viewpoint, PTIME algorithms have been proposed to solve these two specifications of probabilistic diagnosability [4]. Predictability in pLTS with arbitrary small probability of erroneous information has also been studied in [5].
In case a system is not diagnosable, one may be able to control it, by forbidding some controllable actions, so that is becomes diagnosable. This property of *active diagnosability* has been studied for probabilistic systems in [1] pursuing the work of [11, 7] for discrete-event systems. Decidability and complexity issues are considered and optimal size diagnosers are synthesized. Interestingly, the diagnosability notion from [1] slightly differs from the original one in [13].

Remaining issues. Some issues remained untouched in the above line of work. First, diagnosability was only considered w.r.t. finite faulty runs. It seems as important to consider diagnosability of correct runs, and ambiguity can also be defined for infinite computations. Second, in most work, the complexity of the varied diagnosability problems and of the diagnosers synthesis were left open. Moreover, optimizing the delay between the fault occurrence and its detection is an important issue. Yet the search for diagnosers (or predictors) with optimal reactivity was not even considered. Last predictability and diagnosability were independently studied while combining them is obviously a fruitful direction.

Contributions. In this paper, we address the above mentioned gaps, and revisit diagnosability and predictability for probabilistic systems, from a semantical as well as a computational perspectives.

- In constrast to existing work, we define diagnosability directly on the ambiguity triggered by the behaviours of the system, and then establish that it is equivalent to the existence of a diagnoser.
- In order to give a firm semantical classification of diagnosability notions, we define criteria for diagnosability in probabilistic systems, depending on (1) whether the ambiguity is related to faulty runs only or to all runs and, (2) whether ambiguity is defined at the level of infinite runs, or for longer and longer finite subruns. A priori these two dimensions yield four specifications. We prove that two of them coincide leading to three main specifications: FF-diagnosability, IA-diagnosability used in [1] and FA-diagnosability, and

we establish the connections between them. In addition we show that FF-diagnosability is equivalent to the A-diagnosability of [13] for finite pLTS and that this hypothesis is necessary.

- For finite state probabilistic systems, we show that these three notions of diagnosability can be characterized based on deterministic (finite or Büchi) automata acting as *monitors*, and synchronized with the pLTS. We further prove that the diagnosability problem (for all three specifications) is PSPACE-complete, contradicting the polynomial time result for FF-diagnosability [4], and identify the error in their algorithm.
- Afterwards, we design algorithms for the synthesis of finite-memory diagnosers and prove that their size $2^{\Theta(n)}$ (where n is the number of states of the pLTS model) is optimal.
- Since predictability is an interesting alternative to diagnosability, we introduce two possible specifications for predictability in probabilistic systems, and show that in both cases the predictability problem is NLOGSPACE-complete. Yet, as for diagnosers, the optimal size of predictors is in $2^{\Theta(n)}$.
- Last, we introduce and study *prediagnosability* that combines the benefits of predictability and diagnosability: depending on the observations, a prediagnoser behaves as a diagnoser or a predictor. Prediagnosability is of interest since predictability is more difficult to achieve than diagnosability, also prediagnosers can be seen as "as soon as possible" diagnosers. For the varied notions of prediagnosability, we establish that the prediagnosability problem is PSPACE-complete and design prediagnosers with optimal size.
- Summarizing we provide a full picture of the hierarchy for the different notions and the frontier between NLOGSPACE and PSPACE-complete problems.

Organization. In Section 2, we introduce probabilistic LTS, define the possible diagnosability specifications, establish their connection. In Section 3, we provide characterizations for diagnosability of finite pLTS and we determine the exact complexity of the diagnosability problems. In Section 4, we design algorithms for synthesis of diagnosers with optimal size. In Section 5, we study predictability and prediagnosis, and focus on optimal diagnosers. All the proofs and additional results can be found in the companion research report [2].

2 Diagnosability specification

In the context of stochastic discrete event systems diagnosis, systems are often modeled using labeled transition systems.

▶ **Definition 1.** A *probabilistic labeled transition system* (pLTS) is a tuple $\mathcal{A} = \langle Q, q_0, \Sigma, T, \mathbf{P} \rangle$ where:

- Q is a set of states with $q_0 \in Q$ the initial state;
- Σ is a finite set of events;
- $T \subseteq Q \times \Sigma \times Q$ is a set of transitions;
- $\mathbf{P} : T \to \mathbb{Q}_{>0}$ is the probabilistic transition function fulfilling for all $q \in Q$: $\sum_{(q,a,q') \in T} \mathbf{P}(q, a, q') = 1$.

Observe that a pLTS is a labeled transition system (LTS) equipped with transition probabilities. The transition relation of the underlying LTS is defined by: $q \xrightarrow{a} q'$ for $(q, a, q') \in T$; this transition is then said to be *enabled* in q. A pLTS is said to be *live* if in every state q of the pLTS, a transition is enabled. We assume the pLTS we consider are countably branching, *i.e.*, in every state q, only countably many transitions are enabled, so that the summation $\sum_{(q,a,q') \in T} \mathbf{P}(q, a, q')$ is well-defined.

Let us now introduce some important notions and notations that will be used throughout the paper. A *run* ρ of a pLTS $\mathcal{A}$ is a (finite or infinite) sequence $\rho = q_0 a_0 q_1 \dots$ such that for all i, $q_i \in Q$, $a_i \in \Sigma$ and when q_{i+1} is defined, $q_i \xrightarrow{a_i} q_{i+1}$. The notion of run can be generalized, starting from an arbitrary state q. We write Ω for the set of all infinite runs of $\mathcal{A}$ starting from q_0, assuming the pLTS is clear from context. When it is finite, ρ ends in a state q and its *length*, denoted $|\rho|$, is the number of actions occurring in it. Given a finite run $\rho = q_0 a_0 q_1 \dots q_n$ and a (finite or infinite) run $\rho' = q_n a_n q_{n+1} \dots$, we call concatenation of ρ and ρ' and we write $\rho\rho'$ the run $q_0 a_0 q_1 \dots q_n a_n q_{n+1} \dots$; the run ρ is then a *prefix* of $\rho\rho'$, which we denote $\rho \preceq \rho\rho'$. The *cylinder* defined by a finite run ρ is the set of all infinite runs that extend ρ: $C(\rho) = \{\rho' \in \Omega \mid \rho \preceq \rho'\}$. The sequence associated with $\rho = q a_0 q_1 \dots$ is the word $\sigma_\rho = a_0 a_1 \dots$, and we write equally $q \overset{\rho}{\Rightarrow}$ or $q \overset{\sigma_\rho}{\Longrightarrow}$ (resp. $q \overset{\rho}{\Rightarrow} q'$ or $q \overset{\sigma_\rho}{\Longrightarrow} q'$) for an infinite (resp. finite) run ρ. A state q is *reachable* (from q_0) if there exists a run such that $q_0 \overset{\rho}{\Rightarrow} q$, which we alternatively write $q_0 \Rightarrow q$. The language of pLTS $\mathcal{A}$ consists of all infinite words that label runs of $\mathcal{A}$ and is formally defined as $\mathcal{L}^\omega(\mathcal{A}) = \{\, \sigma \in \Sigma^\omega \mid q_0 \overset{\sigma}{\Rightarrow} \}$.

Forgetting the labels and merging (and summing the probabilities of) the transitions with same source and target, a pLTS yields a discrete time Markov chain (DTMC). As usual for DTMC, the set of infinite runs of $\mathcal{A}$ is the support of a probability measure defined by Caratheodory's extension theorem from the probabilities of the cylinders:

$$\mathbb{P}(C(q_0 a_0 q_1 \dots q_n)) = \mathbf{P}(q_0, a_1, q_1) \cdots \mathbf{P}(q_{n-1}, a_{n-1}, q_n) \ .$$

In order to formalize problems related to fault diagnosis, we partition the event set Σ into two disjoint sets Σ_o and Σ_u, the sets of *observable* and of *unobservable events*, respectively. Moreover, we distinguish a special *fault* event $\mathbf{f} \in \Sigma_u$. Let $\sigma \in \Sigma^*$ be a finite word; its length is denoted $|\sigma|$. The projection of σ onto Σ_o is defined inductively by: $\mathcal{P}(\varepsilon) = \varepsilon$; for $a \in \Sigma_o$, $\mathcal{P}(\sigma a) = \mathcal{P}(\sigma)a$; and $\mathcal{P}(\sigma a) = \mathcal{P}(\sigma)$ for $a \notin \Sigma_o$. Write $|\sigma|_o$ for $|\mathcal{P}(\sigma)|$. When σ is an infinite word, its projection is the limit of the projections of its finite prefixes. This projection is applicable to runs via their associated sequence; it can be either finite or infinite. As usual the projection is extended to languages. With respect to the partition of $\Sigma = \Sigma_o \uplus \Sigma_u$, a pLTS $\mathcal{A}$ is *convergent* if there is no infinite sequence of unobservable events from any reachable state: $\mathcal{L}^\omega(\mathcal{A}) \cap \Sigma^* \Sigma_u^\omega = \emptyset$. When $\mathcal{A}$ is convergent, for every $\sigma \in \mathcal{L}^\omega(\mathcal{A})$, $\mathcal{P}(\sigma) \in \Sigma_o^\omega$. In the rest of the paper we assume that pLTS are convergent. We will refer to a *sequence* for a finite or infinite word over Σ, and an *observed sequence* for a finite or infinite sequence over Σ_o. Clearly, the projection onto Σ_o of a sequence yields an observed sequence.

The *observable length* of a run ρ denoted $|\rho|_o \in \mathbb{N} \cup \{\infty\}$, is the number of observable actions that occur in it: $|\rho|_o = |\sigma_\rho|_o$. A *signalling* run is a finite run whose last action is observable. Signalling runs are precisely the relevant runs w.r.t. partial observation issues since each observable event provides an additional information about the execution to an external observer. In the sequel, SR denotes the set of signalling runs, and SR_n the set of signalling runs of observable length n. Since we assume that the pLTS are convergent, for all $n > 0$, SR_n is equipped with a probability distribution defined by assigning measure $\mathbb{P}(\rho) = \mathbb{P}(C(\rho))$ to each $\rho \in \mathsf{SR}_n$. Given ρ a finite or infinite run, and $n \leq |\rho|_o$, $\rho_{\downarrow n}$ denotes the signalling subrun of ρ of observable length n. For convenience, we consider the empty run q_0 to be the single signalling run, of null length.

Let $\mathcal{A}$ be a pLTS. A run ρ is *faulty* if σ_ρ contains $\mathbf{f}$, otherwise it is *correct.* W.l.o.g., by considering two copies of each state, we assume that the states of $\mathcal{A}$ are partitioned into correct states and faulty states: $Q = Q_f \uplus Q_c$ where Q_f are faulty states, and Q_c correct states. Faulty (resp. correct) states are only reachable by faulty (resp. correct) runs. An observed sequence $\sigma \in \Sigma_o^\omega$ is *surely correct* if $\mathcal{P}^{-1}(\sigma) \cap \mathcal{L}^\omega(\mathcal{A}) \subseteq (\Sigma \setminus \mathbf{f})^\omega$; it is *surely faulty*

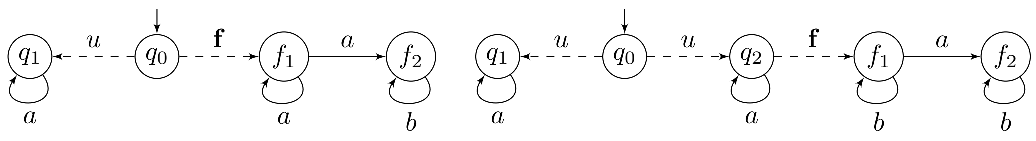


Figure 1 Left: a pLTS that is IF-diagnosable but not IA-diagnosable. Right: a pLTS that is IA-diagnosable but not FA-diagnosable.

if $\mathcal{P}^{-1}(\sigma) \cap \mathcal{L}^\omega(\mathcal{A}) \subseteq \Sigma^*\mathbf{f}\Sigma^\omega$; otherwise, it is *ambiguous*. For finite sequences, we need to rely on signalling runs: a finite observed sequence $\sigma \in \Sigma_o^*$ is *surely faulty* (resp. *surely correct*) if for every signalling run ρ with $\mathcal{P}(\sigma_\rho) = \sigma$, ρ is faulty (resp. correct); otherwise it is ambiguous. A (finite signalling or infinite) run ρ is *surely faulty* (resp. *surely correct*, *ambiguous*) if $\mathcal{P}(\rho)$ is surely faulty (resp. surely correct, ambiguous).

In order to introduce diagnosability, we define different subsets of infinite runs.

▶ **Definition 2** (Ambiguous runs)**.** Let $\mathcal{A}$ be a pLTS and $n \in \mathbb{N}$ with $n \geq 1$. Then:

- FAmb_∞ is the set of infinite faulty ambiguous runs of $\mathcal{A}$;
- CAmb_∞ is the set of infinite correct ambiguous runs of $\mathcal{A}$;
- FAmb_n is the set of infinite runs of $\mathcal{A}$ whose signalling subrun of observable length n is faulty and ambiguous;
- CAmb_n is the set of infinite runs of $\mathcal{A}$ whose signalling subrun of observable length n is correct and ambiguous.

We propose four possible specifications of diagnosability for probabilistic systems. There are two discriminating criteria: whether the non ambiguity requirement holds for faulty runs only (_F) or for all runs (_A), and whether ambiguity is defined at the infinite run level (I_) or for longer and longer finite signalling subruns (F_).

▶ **Definition 3** (Diagnosability specifications)**.** Let $\mathcal{A}$ be a pLTS. Then:

- A pLTS $\mathcal{A}$ is IF-diagnosable if $\mathbb{P}(\mathsf{FAmb}_\infty) = 0$.
- A pLTS $\mathcal{A}$ is IA-diagnosable if $\mathbb{P}(\mathsf{FAmb}_\infty \uplus \mathsf{CAmb}_\infty) = 0$.
- A pLTS $\mathcal{A}$ is FF-diagnosable if $\limsup_{n\to\infty} \mathbb{P}(\mathsf{FAmb}_n) = 0$.
- A pLTS $\mathcal{A}$ is FA-diagnosable if $\limsup_{n\to\infty} \mathbb{P}(\mathsf{FAmb}_n \uplus \mathsf{CAmb}_n) = 0$.

Let us illustrate these specifications on the two pLTS of Figure 1 where $\{u, \mathbf{f}\}$ is the set of unobservable events, represented by dashed arrows. Here and later on, unless mentioned, the transitions outgoing a state are uniformly distributed. On the left, a faulty run will almost surely produce a b-event that cannot be mimicked by the single correct run. Thus this pLTS is IF–diagnosable. The unique correct run $\rho = q_0uq_1(aq_1)^\omega$ has probability $\frac{1}{2}$ and its corresponding observed sequence a^ω is ambiguous. Thus this pLTS is not IA-diagnosable. On the right, any infinite faulty run will contain a b-event, and cannot be mimicked by a correct run, therefore $\mathsf{FAmb}_\infty = \emptyset$. The two infinite correct runs have a^ω as observed sequence, and cannot be mimicked by a faulty run, thus $\mathsf{CAmb}_\infty = \emptyset$. As a consequence, this pLTS is IA-diagnosable. Consider now the infinite correct run $\rho = q_0uq_1(aq_1)^\omega$. It has probability $\frac{1}{2}$, and all its finite signalling subruns are ambiguous since their observed sequence is a^n, for some $n \in \mathbb{N}$. Thus for all $n \geq 1$, $\mathbb{P}(\mathsf{CAmb}_n) \geq \frac{1}{2}$, so that this pLTS is not FA-diagnosable.

The next theorem establishes the connections between these definitions.

▶ **Theorem 4.** *The different diagnosability notions for pLTS relate according to the table below. Moreover, all implications hold for infinite-state pLTS, and non implications already hold for finite-state pLTS. (The implication marked with * requires finitely branching pLTS.)*

Diagnosability	*All runs*		*Faulty runs*
Signalling runs	FA	$\Rightarrow$ $\not\Leftarrow$	FF
	$\Downarrow\not\Uparrow$		$\Downarrow\Uparrow^*$
Infinite runs	IA	$\Rightarrow$ $\not\Leftarrow$	IF

To conclude this section, we compare IF-diagnosability with A-diagnosability from [13].

▶ **Theorem 5.** *A finite pLTS $\mathcal{A}$ is* IF*-diagnosable if and only if it is A-diagnosable, that is:* $\forall\varepsilon > 0$, $\exists N_\varepsilon \in \mathbb{N}$, *for every faulty signalling run* ρ *and every* $n \geq N_\varepsilon$, $\mathbb{P}(\{\rho' \in \mathsf{FAmb}_{n+|\rho|_o} \mid \rho \preceq \rho'\}) < \varepsilon\mathbb{P}(\rho)$. *This condition is only sufficient for finitely branching infinite pLTS.*

3 Complexity of diagnosability

In this section, we establish the complexity of diagnosability stated in the next theorem.

▶ **Theorem 6.** *The* IF*-diagnosability,* IA*-diagnosability, and* FA*-diagnosability problems for finite pLTS are* PSPACE*-complete.*

To prove membership in PSPACE, we provide characterizations of the different diagnosability notions we introduced. For each notion of diagnosability, we proceed similarly. First, given a pLTS $\mathcal{A}$ we design a deterministic automaton that accepts some (finite or infinite) observed sequences of $\mathcal{A}$. Then we build the synchronized product of this automaton with $\mathcal{A}$, to obtain another pLTS with the same stochastic behaviour as $\mathcal{A}$ but augmented with additional information about the current run, that will be useful for diagnosability. Finally, we characterize diagnosability by graph properties on the synchronized product.

Here we only detail the procedure for IA-diagnosability. Its automaton $\mathsf{IA}(\mathcal{A})$ is the deterministic Büchi automaton introduced in [7]. Its states are triples of disjoint subsets of states (U, V, W) where given some observed sequence, U is the set of possible correct states and V and W are possible faulty states. The decomposition between V and W reflects the fact that the IA-automaton tries to resolve the ambiguity between U and W (when both are non empty), while V corresponds to a waiting room of states reached by faulty runs that will be examined when the current ambiguity is resolved. The set F of accepting states consists of all triples (U, V, W) with $U = \emptyset$ or $W = \emptyset$. When $U = \emptyset$, the current signalling run is surely faulty. When $W = \emptyset$ the current signalling run may be ambiguous (if $V \neq \emptyset$) but the "oldest" possible faulty runs have been discarded. Hence, any infinite observed sequence of $\mathcal{A}$ passing infinitely often through F is not ambiguous (ambiguities are resolved one after another).

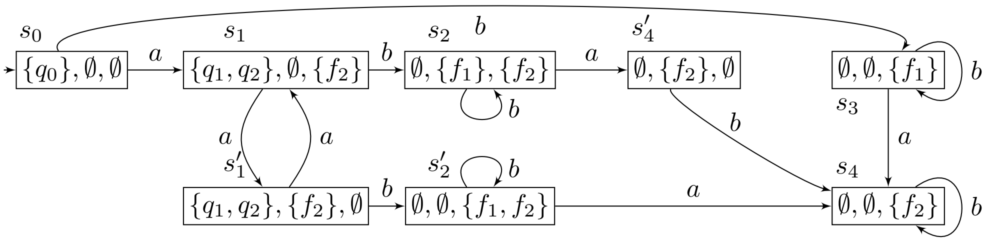
Figure 2 shows the IA-automaton of the pLTS depicted on the right of Figure 1.

Observe that, despite the fact that all observed sequences a^n are ambiguous as witnessed by the possible faulty state f_2, a^ω, which is indeed unambiguous, is accepted by the IA-automaton since its execution infinitely often visits state $(\{q_1, q_2\}, \{f_2\}, \emptyset)$.

To come up with a characterization, one builds $\mathcal{A}_{\mathsf{IA}} = \mathcal{A} \times \mathsf{IA}(\mathcal{A})$, the product of $\mathcal{A}$ and $\mathsf{IA}(\mathcal{A})$ synchronized over observed events.

▶ **Proposition 7.** *A finite pLTS $\mathcal{A}$ is* IA*-diagnosable if and only if* $\mathcal{A}_{\mathsf{IA}}$ *has no bottom strongly connected component (BSCC) such that:*

- *either, all its states* (q, U, V, W) *fulfill* $q \in Q_f$ *and* $U \neq \emptyset$;
- *or all its states* (q, U, V, W) *fulfill* $q \in Q_c$ *and* $W \neq \emptyset$.



s_0
$\{q_0\}, \emptyset, \emptyset$
a
s_1
$\{q_1, q_2\}, \emptyset, \{f_2\}$
b
s_2
$\emptyset, \{f_1\}, \{f_2\}$
b
a
s'_4
$\emptyset, \{f_2\}, \emptyset$
b
$\emptyset, \emptyset, \{f_1\}$
b
s_3
a
a
a
s'_1
$\{q_1, q_2\}, \{f_2\}, \emptyset$
b
s'_2
$\emptyset, \emptyset, \{f_1, f_2\}$
b
a
b
s_4
$\emptyset, \emptyset, \{f_2\}$
b

Figure 2 The IA-automaton of pLTS depicted on the right of Figure 1.

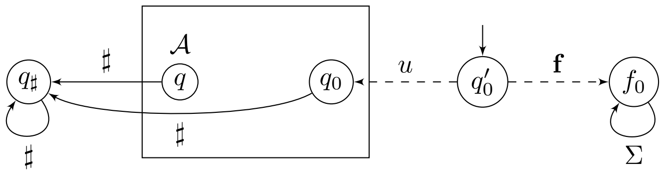


Figure 3 A reduction for PSPACE-hardness of IA-diagnosability.

The decision algorithm for IA-diagnosability checks whether the above characterization is satisfied by looking for a state that violates the disjunction and then checking that it belongs to a BSCC. This can be done in polynomial space without explicitly building $\mathcal{A}_{\mathsf{IA}}$, and relying on Savitch's theorem.

In order to establish a lower bound for the complexity of IA-diagnosability, we introduce a variant of language universality. A language $\mathcal{L}$ over an alphabet Σ is said *eventually universal* if there exists a word $v \in \Sigma^*$ such that $v^{-1}\mathcal{L} = \Sigma^*$, where $v^{-1}\mathcal{L}$ denotes the left quotient of $\mathcal{L}$ by v: $v^{-1}\mathcal{L} = \{v' \mid vv' \in \mathcal{L}\}$. Recently, several variants of the universality problem were shown to be PSPACE-complete [10] but, to the best of our knowledge, eventual universality has not yet been considered.

Because of our diagnosis framework, we focus on live non deterministic finite automata (NFA). Similarly to pLTS, an NFA is *live* if from every state there is at least one outgoing transition. The language of an NFA $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is defined as the set of finite words that are accepted by $\mathcal{A}$. We reduce the universality problem for NFA, which is known to be PSPACE-complete [9] to the eventual universality problem to obtain the following result.

▶ **Proposition 8.** *Let $\mathcal{A}$ be a live NFA where all states are terminal. Then deciding whether $\mathcal{L}(\mathcal{A})$ is eventually universal is* PSPACE*-hard.*

Let us sketch how we reduce this problem to IA-diagnosability. Given a live NFA $\mathcal{A}$ over Σ where all states are terminal, one builds the pLTS of Figure 3 where $\Sigma \cup \{\sharp\}$ are observable. Since a correct run almost surely "outputs" a $\sharp$, ambiguity may only occur with faulty runs. Since after the fault one observes Σ^*, using the characterization in Proposition 7 one concludes that the pLTS is not IA-diagnosable if and only if $\mathcal{A}$ is eventually universal.

Theorem 6 seems to contradict the PTIME decision procedure from [4] for A-diagnosability (or, equivalently IF-diagnosability). However, we establish that:

▶ **Fact 9.** *The* PTIME *algorithm of [4] for A-diagnosability is erroneous.*

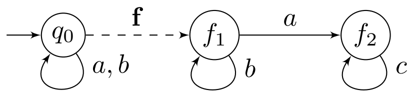


Figure 4 A pLTS which is IA-diagnosable.

4 Diagnoser construction

In this section, we focus on the construction of diagnosers. A diagnoser is a function $D : \Sigma_o^* \to \{?, \top, \bot\}$ assigning to every finite observation sequence a verdict. Informally when a diagnoser outputs ? it does not provide any information, while $\top$ ensures that a fault is certain and $\bot$ that some information about correctness has been provided. We consider the natural partial order $\prec$ on these values defined by $? \prec \top$ and $? \prec \bot$.

A finite memory diagnoser is given by a tuple $(M, \Sigma_o, m_0, \mathsf{up}, D_{fm})$ where M is a finite set of memory states, $m_0 \in M$ is the initial memory state, $\mathsf{up} : M \times \Sigma_o \to M$ is a memory update function, and finally $D_{fm} : M \to \{?, \top, \bot\}$ is a diagnoser function. The mapping up is extended into a function $\mathsf{up} : M \times \Sigma_o^* \to M$ defined inductively by $\mathsf{up}(m, \varepsilon) = m$ and $\mathsf{up}(m, wa) = \mathsf{up}(\mathsf{up}(m, w), a)$. A finite memory diagnoser is not a diagnoser as defined above, yet it induces the diagnoser defined by $D(w) = D_{fm}(\mathsf{up}(m_0, w))$.

Diagnosers we define in the sequel will have two important properties: soundness and reactivity. Soundness ensures that the information provided is accurate and reactivity specifies which pieces of information the diagnoser must provide. The precise soundness and reactivity requirements depend on the diagnosability notion of interest. Moreover, we restrict to diagnosers that, once they output $\top$, never change their verdict in the future. Note that any sound diagnoser can be turned into one that is sound and satisfies this commitment property. In this short version, we only introduce IA-diagnosers (the synthesis of FA-diagnosers and IF-diagnosers is similar and even simpler). Intuitively, IA-diagnosers may resolve an ambiguity late, while another one has already been produced.

▶ **Definition 10.** An IA-diagnoser for $\mathcal{A}$ is a function $D : \Sigma_o^* \to \{\top, \bot, ?\}$ such that

soundness For all $w \in \Sigma_o^*$
- if $D(w) = \top$, then w is surely faulty;
- if $D(w) = \bot$, letting $|D(w)|_\bot = |\{0 < n \leq |w| \mid D(w_{\leq n}) = \bot\}|$, then for all signalling run ρ such that $\mathcal{P}(\rho) = w$, $\rho_{\downarrow |D(w)|_\bot}$ is correct.

reactivity $\mathbb{P}(\{\rho \in \Omega \mid D_{\sup}(\mathcal{P}(\rho)) = ?\}) = 0$ where for $w \in \Sigma_o^\omega$, $D_{\sup}(w) = \limsup_{n \to \infty} D(w_{\leq n})$.
($D_{\sup}$ is well-defined since once the diagnoser outputs $\top$, it always sticks to this verdict.)

The reactivity condition requires that almost surely the diagnoser detects a fault or guarantees that longer and longer subruns of the current run are correct. Soundness of $\top$ verdict implies that indeed the run is faulty. The interpretation of $D(w) = \bot$ is that the diagnoser ensures that any signalling subrun of length $|D(w)|_\bot \leq |w|$ of a signalling run for w is correct. Of course it may deduce this information from the last $|w| - |D(w)|_\bot$ observations. This is illustrated on the example of Figure 4 for which we describe an IA-diagnoser. After observing any sequence $wbaa$, with $w \in \{a, b\}^*$, the diagnoser knows a posteriori that two steps before, that is after the observation of wb, the run was necessarily correct. Indeed, observing the suffix aa is not possible after a fault, yet wba is not surely correct. The function D defined by: for $w \in \{a, b\}^*(ab + aa)$, $D(w) = \bot$, for $w \in \{a, b, c\}^*c$, $D(w) = \top$ and otherwise $D(w) =?$, is an IA-diagnoser.

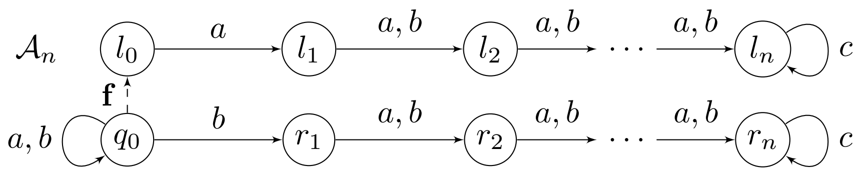


Figure 5 Example of an IA-diagnosable pLTS requiring an IA-diagnoser with exponential size.

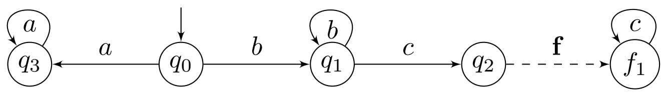


Figure 6 A 0-surely predictable and 1-predictable pLTS.

The next proposition establishes that this definition of diagnosers is appropriate for IA-diagnosability. Furthermore it provides tight lower and upper bounds for the size of IA-diagnosers. The pLTS of Figure 5 is used to prove the lower bound. Intuitively, every IA-diagnoser for this pLTS must decide, on observing a c, whether the run is faulty or correct. To do so, it must remember whether, n observations earlier, the event was a or b. Due to the self-loop on q_0, it cannot know when a c will occur, and must remenber the n last observations. This requires at least 2^n memory states.

▶ **Proposition 11.** *A finite pLTS $\mathcal{A}$ is* IA*-diagnosable if and only if it admits an* IA*-diagnoser. For every pLTS $\mathcal{A}$ with n_c correct states and n_f faulty states which is* IA*-diagnosable, one can build an* IA*-diagnoser with at most $2^{n_c}3^{n_f}$ states. There is a family $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ of* IA*-diagnosable pLTS such that $\mathcal{A}_n$ has $2n+2$ states and it admits no* IA*-diagnoser with less than 2^n memory states.*

5 Predictability and prediagnosis

Predictability. Fault predictability has been first introduced for LTS in [6]: in words, an LTS is predictable (resp. k-predictable) if a fault can be predicted (resp. at least before k observations) whatever the future behavior of the LTS. There are two possible adaptations for pLTS: (1) either one sticks to the original definition and requires that the fault surely occurs or, (2) one relaxes it and only requires that the fault almost surely occurs.

In order to reason about predictability, we introduce some particular prefixes of a run. For a finite run ρ, and $k \in \mathbb{N}$, we define $pre_k(\rho)$, the k-past of ρ, by $pre_k(\rho) = \rho_{\downarrow|\rho|_o-\min(k,|\rho|_o)}$. For example, in the pLTS of Figure 6, $pre_0(q_0bq_1\mathbf{f}q_2) = q_0bq_1$ as $\mathbf{f}$ is unobservable and $pre_1(q_0bq_1\mathbf{f}q_2) = q_0$. In fact for $k \geq 1$, $pre_k(q_0bq_1\mathbf{f}q_2) = q_0$.

We also introduce sets of observed sequences defined on their possible future behaviors. In words, an observed sequence σ forbids prediction of a fault when there is still either a correct infinite run where σ is a prefix of its observed sequence (UPC) or a set of positive measure of such runs (UPSC). Thus in order to be k-predictable, k observations before a possible fault the observed sequence should not belong to these sets (see Definition 13).

▶ **Definition 12** (ultimately possibly (significantly) correct)**.** Let σ be a finite observed sequence of a pLTS $\mathcal{A}$. Then:

- σ is *ultimately possibly correct* if $\{\rho' \in \Omega \mid \sigma \preceq \mathcal{P}(\rho')\} \cap \mathsf{C}_\infty \neq \emptyset$. The set of ultimately possibly correct observed sequences is denoted UPC.
- σ is *ultimately possibly significantly correct* if $\mathbb{P}(\{\rho' \in \Omega \mid \sigma \preceq \mathcal{P}(\rho')\} \cap \mathsf{C}_\infty) > 0$. The set of ultimately possibly significantly correct observed sequences is denoted UPSC.

▶ **Definition 13** ((sure) predictability)**.** Let $k \in \mathbb{N}$.
- A pLTS $\mathcal{A}$ is *k-surely predictable* if for every run $\rho\mathbf{f}q$ of $\mathcal{A}$, $\mathcal{P}(pre_k(\rho)) \notin \mathsf{UPC}$;
- A pLTS $\mathcal{A}$ is *k-predictable* if for every run $\rho\mathbf{f}q$ of $\mathcal{A}$, $\mathcal{P}(pre_k(\rho)) \notin \mathsf{UPSC}$.

Observe that in the previous definition, one can safely restrict to check the condition on correct runs ρ by considering the first occurrence of a fault in the run $\rho\mathbf{f}q$.

For example, the pLTS of Figure 6 is 0-surely predictable. Every correct run ρ that is followed by $\mathbf{f}$ is such that $\mathcal{P}(\rho) = b^n c$ for some $n \geq 1$. As it is the unique signalling run with such an observed sequence, the fault can be predicted. It is not 1-surely predictable as the 1-past of $\rho = q_0 b q_1 c q_2 \mathbf{f} f_1$ is $pre_1(\rho) = q_0 b q_1$ and the infinite run $\rho' = q_0(bq_1)^\omega$ is correct. However it is 1-predictable as for every signalling run with observed sequence b^n for some $n \geq 1$ (thus ending in q_1) a fault eventually almost surely occurs. Finally it is not 2-predictable since the 2-past of $\rho = q_0 b q_1 c q_2 \mathbf{f} f_1$ is q_0 and the infinite correct run $\rho = q_0(aq_3)^\omega$ has probability $\frac{1}{2}$.

We have established all the relations between the different notions of diagnosability and predicatibility (see Figure 8 in the conclusion). The main result about predicatibility is given in the next theorem, and highlights the complexity gap between predictability and diagnosability for probabilistic systems (recall that their complexity coincide for LTS). Despite this difference, the size of optimal predictors is comparable to the one of optimal diagnosers (see details in our research report [2]).

▶ **Theorem 14.** *Deciding, given $\mathcal{A}$ a pLTS and $k \in \mathbb{N}$, whether $\mathcal{A}$ is k-predictable (resp. surely k-predictable) is an* NLOGSPACE*-complete problem. Moreover, the same complexity applies assuming k is fixed (rather than given as input).*

Prediagnosis. On the one hand, diagnosis is concerned with detection of faults that have occurred: given a sequence of observations a diagnoser tries to detect that a fault has occurred in the *past* of all consistent behaviors. On the other hand, prediction is concerned with anticipation of faults: given a sequence of observations a predictor tries to detect that a fault will eventually occur in the *future* of all consistent behaviors. The notion we introduce now, *prediagnosis*, concerns detection of faults both in the past and in the future.

Let us start by introducing two sets of infinite faulty runs that make prediagnosis impossible. FUPC_∞ is the set of faulty runs that admit for all their finite prefixes a compatible infinite correct run. The condition is strengthened for FUPSC_∞ which gathers the faulty runs that admit for all their finite prefixes, a positive measure of compatible infinite correct runs.

▶ **Definition 15.** Let $\mathcal{A}$ be a pLTS. Then:
- FUPC_∞, the set of *faulty, ultimately possibly correct* runs is defined by:
  $\mathsf{FUPC}_\infty = \{\rho \in \Omega \mid \rho \text{ faulty and } \forall i \in \mathbb{N},\ \mathcal{P}(\rho_{\downarrow i}) \in \mathsf{UPC}\}$
- FUPSC_∞, the set of *faulty, ultimately possibly significantly correct* runs is defined by:
  $\mathsf{FUPSC}_\infty = \{\rho \in \Omega \mid \rho \text{ faulty and } \forall i \in \mathbb{N},\ \mathcal{P}(\rho_{\downarrow i}) \in \mathsf{UPSC}\}$

The reactivity requirement for prediagnosers will impose that these sets are negligible. The difference between these two sets impacts correctness: relying on FUPC_∞ provides a *sure* correctness while relying on FUPSC_∞ only provides an *almost sure* correctness.

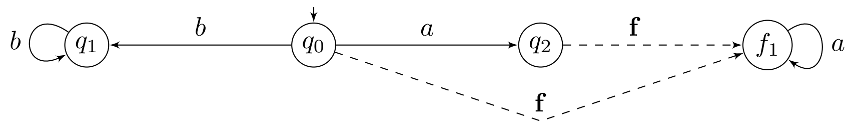


Figure 7 A non-predictable pLTS, for which a sure prediagnoser is quicker than all diagnosers.

▶ **Definition 16** ((Sure) Prediagnosability)**.** Let $\mathcal{A}$ be a pLTS. Then:

- $\mathcal{A}$ is *surely prediagnosable* if $\mathbb{P}(\mathsf{FUPC}_\infty)=0$;
- $\mathcal{A}$ is *prediagnosable* if $\mathbb{P}(\mathsf{FUPSC}_\infty)=0$.

Surprisingly, sure prediagnosability lies strictly between FF-diagnosability and IF-diagnosability with equivalence for finitely branching pLTS. Also (sure) 0-predictability implies (sure) prediagnosability. As expected, the less demanding specification is prediagnosability. All the relations that we have established between the different notions of diagnosability, predictability and prediagnosability are described by Figure 8 in the conclusion. From a complexity point of view, prediagnosability is equivalent to diagnosability:

▶ **Theorem 17.** *The (sure) prediagnosability problem is* PSPACE*-complete.*

In the research report [2], we formally define and study the notion of *prediagnosers*. Here we informally discuss the interest of prediagnosers. While sure prediagnosability and IF-diagnosability are equivalent for finitely branching pLTS, there are differences between sure prediagnosers and IF-diagnosers. An IF-diagnoser is a sure prediagnoser, but a sure prediagnoser may output a verdict $\top$ even before a fault. This phenomenon occurs even if the pLTS is non predictable. The non predictable pLTS of Figure 7 points out this difference. A diagnoser may output $\top$ only after observing two a's, since then, surely a fault occurred. In contrast, a sure prediagnoser can already output $\top$ after observing the first a. In fact this pLTS is FA-diagnosable since after an occurrence of b, the run is surely correct. Prediagnosers, can be thought of as monitors that emit verdicts as soon as possible, while preserving soundness. In the proof that prediagnosability is equivalent to the existence of a prediagnoser, the prediagnosers we construct are indeed optimal in that sense.

6 Conclusion

In this work, we settled the foundations of diagnosability and predictability for partially observed stochastic systems. In particular, we investigated semantical issues and provided several meaningful definitions for diagnosability and predictability in a probabilistic context. We also introduced prediagnosability, that combines the advantages of diagnosability and predictability. Beyond providing relations between these notions, we obtained tight complexity bounds using graph-based characterizations on the product of the system under scrutiny and an appropriate monitor. The complexity ranges from NLOGSPACE-completeness for predictability to PSPACE-completeness for diagnosability and prediagnosability, as summarized on Figure 8. Last, we proved exponential almost matching lower and upper bounds for the diagnosers, predictors, and prediagnosers synthesis problems.

The present contribution opens several interesting research perspectives. First of all, the decidability status (and in the positive case, the precise complexity) of the approximate diagnosability (AA-diagnosability) introduced in [13] is still open since we only proved the algorithm from [4] to be erroneous (see [2]). Second, beyond diagnosability and its

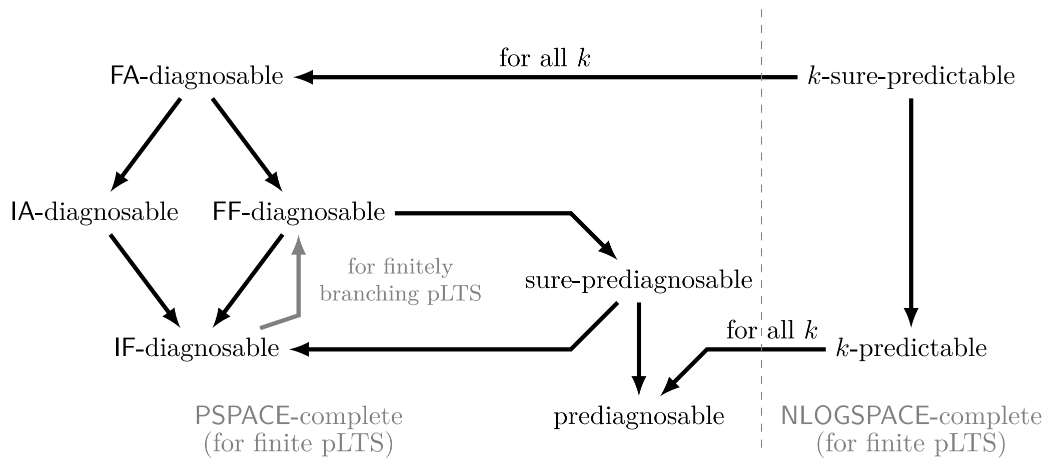


Figure 8 Summarizing relations between specifications, and associated complexities.

variants (predictability and prediagnosability), we wish to conduct a systematic study of other paradigms related to partial observability, such as opacity or detectability, in a probabilistic context. Last, we plan to move to more quantitative versions of diagnosis including optimization issues. The objective would be to minimize the observational capacities of the monitor, either spatially or timely by restricting either the observable actions, or the observation time instants, while preserving diagnosability.

References

1 N. Bertrand, E. Fabre, S. Haar, S. Haddad, and L. Hélouët. Active diagnosis for probabilistic systems. In *Proceedings of FoSSaCS'14*, volume 8412 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2014.

2 N. Bertrand, S. Haddad, and E. Lefaucheux. Foundation of diagnosis and predictability in probabilistic systems. Research Report LSV-14-09, ENS Cachan, June 2014. Available at `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2014-09.pdf`.

3 B. G. Buchanan and E. H. Shortliffe. *Rule Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1984.

4 J. Chen and R. Kumar. Polynomial test for stochastic diagnosability of discrete-event systems. *IEEE Transactions on Automation Science and Engineering*, 10(4):969–979, 2013.

5 J. Chen and R. Kumar. Failure prognosability of stochastic discrete event systems. In *Proceedings of ACC'14*, pages 2041–2046. IEEE, 2014.

6 S. Genc and S. Lafortune. Predictability of event occurrences in partially-observed discrete-event systems. *Automatica*, 45(2):301–311, 2009.

7 S. Haar, S. Haddad, T. Melliti, and S. Schwoon. Optimal constructions for active diagnosis. In *Proceedings of FSTTCS'13*, volume 24 of *LIPIcs*, pages 527–539. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

8 S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, 2001.

9 A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of SWAT'72*, pages 125–129. IEEE Computer Society, 1972.

10 N. Rampersad, J. Shallit, and Z. Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundamenta Informaticae*, 116(1-4):223–236, 2012.
11 M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, 43(7):908–929, 1998.
12 M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.
13 D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50(4):476–492, 2005.

Lipschitz Robustness of Finite-state Transducers*

Thomas A. Henzinger, Jan Otop and Roopsha Samanta

IST Austria
`{tah,jotop,rsamanta}@ist.ac.at`

Abstract

We investigate the problem of checking if a finite-state transducer is *robust* to uncertainty in its input. Our notion of robustness is based on the analytic notion of Lipschitz continuity – a transducer is K-(Lipschitz) robust if the perturbation in its output is at most K times the perturbation in its input. We quantify input and output perturbation using *similarity functions*. We show that K-robustness is undecidable even for deterministic transducers. We identify a class of functional transducers, which admits a polynomial time automata-theoretic decision procedure for K-robustness. This class includes Mealy machines and functional letter-to-letter transducers. We also study K-robustness of nondeterministic transducers. Since a nondeterministic transducer generates a set of output words for each input word, we quantify output perturbation using *set-similarity functions*. We show that K-robustness of nondeterministic transducers is undecidable, even for letter-to-letter transducers. We identify a class of set-similarity functions which admit decidable K-robustness of letter-to-letter transducers.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.1.1 Models of Computation

Keywords and phrases Robustness; Transducers; Weighted Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.431

1 Introduction

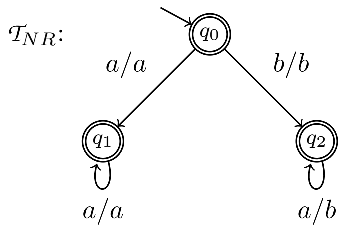
Most computational systems today are embedded in a physical environment. The data processed by such real-world computational systems is often noisy or uncertain. For instance, the data generated by sensors in reactive systems such as avionics software may be corrupted, keywords processed by text processors may be wrongly spelt, the DNA strings processed in computational biology may be incorrectly sequenced, and so on. In the presence of such input uncertainty, it is not enough for a computational system to be functionally correct. An additional desirable property is that of *continuity* or *robustness* — the system behaviour degrades smoothly in the presence of input disturbances [14].

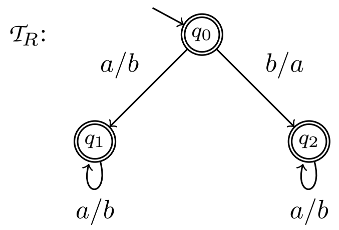
Well-established areas within control theory, such as *robust control* [16], extensively study robustness of systems. However, their results typically involve reasoning about continuous state-spaces and are not directly applicable to inherently discontinuous discrete computational systems. Moreover, uncertainty in robust control refers to differences between a system's model and the actual system; thus robust control focuses on designing controllers that function properly in the presence of perturbation in various internal parameters of a system's model. Given the above, formal reasoning about robustness of computational systems under input uncertainty is a problem of practical as well as conceptual importance.

* This work was supported in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM) and by the Austrian Science Fund (FWF) NFN project S11402-N23 (RiSE).

In our work, we focus on robustness of finite-state transducers, processing finite or infinite words, in the presence of uncertain inputs. Transducers are popular models of input-output computational systems operating in the real world [13, 20, 3, 24]. While many decision problems about transducers have been studied thoroughly over the decades [20, 24], their behaviour under uncertain inputs has only been considered recently [22]. In [22], a transducer was defined to be robust if its output changed proportionally to every change in the input *up to a certain threshold*. In practice, it may not always be possible to determine such a bound on the input perturbation. Moreover, the scope of the work in [22] was limited to the robustness problem for *functional* transducers w. r. t. specific distance functions, and did not consider arbitrary nondeterministic transducers or arbitrary *similarity functions*.

In this paper, we formalize robustness of finite-state transducers as Lipschitz continuity. A function is Lipschitz-continuous if its output changes proportionally to *every* change in the input. Given a constant K and similarity functions d_Σ, d_Γ for computing the input, output perturbation, respectively, a functional transducer $\mathcal{T}$ is defined to be K-Lipschitz robust (or simply, K-robust) w. r. t. d_Σ, d_Γ if for all words s, t in the domain of $\mathcal{T}$ with finite $d_\Sigma(s,t)$, $d_\Gamma(\mathcal{T}(s), \mathcal{T}(t)) \leq K d_\Sigma(s,t)$. Let us consider the transducers $\mathcal{T}_{NR}$ and $\mathcal{T}_R$ below. Recall that the Hamming distance between equal length words is the number of positions in which the words differ. Let d_Σ, d_Γ be computed as the Hamming distance for equal-length words, and be ∞ otherwise. Notice that for words a^{k+1}, ba^k in the domain of the Mealy machine $\mathcal{T}_{NR}$, $d_\Sigma(a^{k+1}, ba^k) = 1$ and the distance between the corresponding output words, $d_\Gamma(a^{k+1}, b^{k+1})$, equals $k+1$. Thus, $\mathcal{T}_{NR}$ is not K-robust for any K. On the other hand, the transducer $\mathcal{T}_R$ is 1-robust: for words a^{k+1}, ba^k, we have $d_\Sigma(a^{k+1}, ba^k) = d_\Gamma((b)^{k+1}, a(b)^k) = 1$, and for all other words s, t in the domain of $\mathcal{T}_R$, either $d_\Sigma(s,t) = \infty$ or $d_\Sigma(s,t) = d_\Gamma(\mathcal{T}_R(s), \mathcal{T}_R(t)) = 0$.





While the K-robustness problem is undecidable even for deterministic transducers, we identify interesting classes of finite-state transducers with decidable K-robustness. We first define a class of functional transducers, called synchronized transducers, which admits a polynomial time decision procedure for K-robustness. This class includes Mealy machines and functional letter-to-letter transducers; membership of a functional transducer in this class is decidable in polynomial time. Given similarity functions computable by weighted automata, we reduce the K-robustness problem for synchronized transducers to the emptiness problem for weighted automata.

We extend our decidability results by employing an *isometry approach.* An *isometry* is a transducer, which for all words s, t satisfies $d_\Gamma(\mathcal{T}(s), \mathcal{T}(t)) = d_\Sigma(s,t)$. We observe that if a transducer $\mathcal{T}_2$ can be obtained from a transducer $\mathcal{T}_1$ by applying isometries to the input and output of $\mathcal{T}_1$, then K-robustness of $\mathcal{T}_1$ and $\mathcal{T}_2$ coincide. This observation enables us to reduce K-robustness of various transducers to that of synchronized transducers.

Finally, we study K-robustness of nondeterministic transducers. Since a nondeterministic transducer generates a set of output words for each input word, we quantify output perturbation using *set-similarity functions* and define K-robustness of nondeterministic transducers w. r. t. such set-similarity functions. We show that K-robustness of nondeterministic transducers is undecidable, even for letter-to-letter transducers. We define three classes of set-similarity functions and show decidability of K-robustness of nondeterministic letter-to-letter transducers w. r. t. one class of set-similarity functions.

In what follows, we first present necessary definitions in Sec. 2. We formalize Lipschitz robustness in Sec. 3. In Sec. 4 and Sec. 5, we study the K-robustness problem for functional transducers, showing undecidability of the general problem and presenting two classes with decidable K-robustness. We study K-robustness of arbitrary nondeterministic transducers in Sec. 6, present a discussion of related work in Sec. 7 and conclude in Sec. 8.

2 Preliminaries

In this section, we review definitions of finite-state transducers and weighted automata, and present similarity functions. We use the following notation. We denote input letters by a, b etc., input words by s, t etc., output letters by a', b' etc. and output words by s', t' etc. We denote the concatenation of words s and t by $s \cdot t$, the i^{th} letter of word s by $s[i]$, the subword $s[i] \cdot s[i+1] \cdot \ldots \cdot s[j]$ by $s[i,j]$, the length of the word s by $|s|$, and the empty word and empty letter by ϵ. Note that for an ω-word s, $|s| = \infty$.

Finite-state Transducers. A finite-state transducer (FST) $\mathcal{T}$ is a tuple $(\Sigma, \Gamma, Q, Q_0, E, F)$ where Σ is the input alphabet, Γ is the output alphabet, Q is a finite nonempty set of states, $Q_0 \subseteq Q$ is a set of initial states, $E \subseteq Q \times \Sigma \times \Gamma^* \times Q$ is a set of transitions[1], and F is a set of accepting states.

A run γ of $\mathcal{T}$ on an input word $s = s[1]s[2]\ldots$ is defined in terms of the sequence: (q_0, w'_1), (q_1, w'_2), $\ldots$ where $q_0 \in Q_0$ and for each $i \in \{1, 2, \ldots\}$, $(q_{i-1}, s[i], w'_i, q_i) \in E$. Let $\mathrm{Inf}(\gamma)$ denote the set of states that appear infinitely often along γ. For an FST $\mathcal{T}$ processing ω-words, a run is accepting if $\mathrm{Inf}(\gamma) \cap F \neq \emptyset$ (Büchi acceptance condition). For an FST $\mathcal{T}$ processing finite words, a run γ: (q_0, w'_1), $\ldots$ (q_{n-1}, w'_n), (q_n, ϵ) on input word $s[1]s[2]\ldots s[n]$ is accepting if $q_n \in F$ (final state acceptance condition). The output of $\mathcal{T}$ along a run is the word $w'_1 \cdot w'_2 \cdot \ldots$ if the run is accepting, and is undefined otherwise. The transduction computed by an FST $\mathcal{T}$ processing infinite words (resp., finite words) is the relation $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$ (resp., $[\![\mathcal{T}]\!] \subseteq \Sigma^* \times \Gamma^*$), where $(s, s') \in [\![\mathcal{T}]\!]$ iff there is an accepting run of $\mathcal{T}$ on s with s' as the output along that run. With some abuse of notation, we denote by $[\![\mathcal{T}]\!](s)$ the set $\{t : (s,t) \in [\![\mathcal{T}]\!]\}$. The input language, $\mathrm{dom}(\mathcal{T})$, of $\mathcal{T}$ is the set $\{s : [\![\mathcal{T}]\!](s) \text{ is non-empty}\}$.

An FST $\mathcal{T}$ is called *functional* if the relation $[\![\mathcal{T}]\!]$ is a function. In this case, we use $[\![\mathcal{T}]\!](s)$ to denote the unique output word generated along any accepting run of $\mathcal{T}$ on input word s. Checking if an arbitrary FST is functional can be done in polynomial time [12]. An FST $\mathcal{T}$ is *deterministic* if $\forall q \in Q, a \in \Sigma$: $|\{q' : (q, a, w', q') \in E\}| \leq 1$. An FST $\mathcal{T}$ is a *letter-to-letter* transducer if for every transition of the form $(q, a, w', q') \in E$, $|w'| = 1$. A *Mealy machine* is a deterministic, letter-to-letter transducer, with every state being an accepting state. In what follows, we use transducers and finite-state transducers interchangeably.
Composition of transducers. Consider transducers $\mathcal{T}_1 = (\Sigma, \Delta, Q_1, Q_{1,0}, E_1, F_1)$ and $\mathcal{T}_2 = (\Delta, \Gamma, Q_2, Q_{2,0}, E_2, F_2)$ such that for every $s \in \mathrm{dom}(\mathcal{T}_1)$, $[\![\mathcal{T}]\!](s) \in \mathrm{dom}(\mathcal{T}_2)$. We define $\mathcal{T}_2 \circ \mathcal{T}_1$, the *composition* of $\mathcal{T}_1$ and $\mathcal{T}_2$, as the transducer $(\Sigma, \Gamma, Q_1 \times Q_2, Q_{1,0} \times Q_{2,0}, E, F_1 \times F_2)$, where E is defined as follows: $(\langle q_1, q_2 \rangle, a, w', \langle q'_1, q'_2 \rangle) \in E$ iff $(q_1, a, t', q'_1) \in E_1$ and upon reading t', $\mathcal{T}_2$ generates w' and changes state from q_2 to q'_2, i.e., iff $(q_1, a, t', q'_1) \in E_1$ and

[1] Note that we disallow ϵ-transitions where the transducer can change state without moving the reading head.

there exist $(q_2, t'[1], w'_1, q_2^1)$, $(q_2^1, t'[2], w'_2, q_2^2)$, …, $(q_2^{k-1}, t'[k], w'_k, q'_2) \in E_2$ such that $k = |t'|$ and $w' = w'_1 \cdot w'_2 \cdot \ldots \cdot w'_k$. Observe that if $\mathcal{T}_1, \mathcal{T}_2$ are functional, $\mathcal{T}_2 \circ \mathcal{T}_1$ is functional and $[\![\mathcal{T}_2 \circ \mathcal{T}_1]\!] = [\![\mathcal{T}_2]\!] \circledcirc [\![\mathcal{T}_1]\!]$, where $\circledcirc$ denotes function composition.

Weighted automata. Recall that a finite automaton (with Büchi or final state acceptance) can be expressed as a tuple (Σ, Q, Q_0, E, F), where Σ is the alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $E \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is a set of accepting states. A weighted automaton (WA) is a finite automaton whose transitions are labeled by rational numbers. Formally, a WA $\mathcal{A}$ is a tuple $(\Sigma, Q, Q_0, E, F, c)$ such that (Σ, Q, Q_0, E, F) is a finite automaton and $c : E \mapsto \mathbb{Q}$ is a function labeling the transitions of $\mathcal{A}$. The transition labels are called *weights*.

Recall that a run π of a finite automaton on a word $s = s[1]s[2]\ldots$ is defined as a sequence of states: $q_0, q_1, \ldots$ where $q_0 \in Q_0$ and for each $i \in \{1, 2, \ldots\}$, $(q_{i-1}, s[i], q_i) \in E$. A run π in a finite automaton processing ω-words (resp., finite words) is accepting if it satisfies the Büchi (resp., final state) acceptance condition. The set of accepting runs of an automaton on a word s is denoted $\mathsf{Acc}(s)$. Given a word s, every run π of a WA $\mathcal{A}$ on s defines a sequence $c(\pi) = (c(q_{i-1}, s[i], q_i))_{1 \leq i \leq |s|}$ of weights of successive transitions of $\mathcal{A}$; such a sequence is also referred to as a weighted run. To define the semantics of weighted automata we need to define the value of a run (that combines the sequence of weights of the run into a single value) and the value across runs (that combines values of different runs into a single value). To define values of runs, we consider *value functions* f that assign real numbers to sequences of rational numbers, and refer to a WA with a particular value function f as an f-WA. Thus, the value $f(\pi)$ of a run π of an f-WA $\mathcal{A}$ on a word s equals $f(c(\pi))$. The value of a word s assigned by an f-WA $\mathcal{A}$, denoted $\mathcal{L}_{\mathcal{A}}(s)$, is the *infimum* of the set of values of all accepting runs, i.e., $\mathcal{L}_{\mathcal{A}}(s) = \inf_{\pi \in \mathsf{Acc}(s)} f(\pi)$ (the infimum of an empty set is infinite).

In this paper, we consider the following value functions: (1) the sum function $\textsc{Sum}(\pi) = \sum_{i=1}^{|\pi|} (c(\pi))[i]$, (2) the discounted sum function $\textsc{Disc}_\delta(\pi) = \sum_{i=1}^{|\pi|} \delta^i (c(\pi))[i]$ with $\delta \in (0, 1)$ and (3) the limit-average function $\textsc{LimAvg}(\pi) = \limsup_{k \to \infty} \frac{1}{k} \sum_{i=1}^{k} (c(\pi))[i]$. Note that the limit-average value function cannot be used with finite sequences. We define $\textsc{ValFunc} = \{\textsc{Sum}, \textsc{Disc}_\delta, \textsc{LimAvg}\}$.

A WA $\mathcal{A}$ is *functional* iff for every word s, all accepting runs of $\mathcal{A}$ on s have the same value.

Decision questions. Given an f-WA $\mathcal{A}$ and a threshold λ, the *emptiness* question asks whether *there exists* a word s such that $\mathcal{L}_{\mathcal{A}}(s) < \lambda$ and the *universality* question asks whether *for all* words s we have $\mathcal{L}_{\mathcal{A}}(s) < \lambda$. The following results are known.

▶ **Lemma 1.** (1) *For every $f \in \textsc{ValFunc}$, the emptiness problem is decidable in polynomial time for nondeterministic f-automata [11, 10].* (2) *The universality problem is undecidable for $\textsc{Sum}$-automata with weights drawn from $\{-1, 0, 1\}$ [17, 1].*

▶ Remark. Weighted automata have been defined over semirings [10] as well as using value functions (along with infimum or supremum) as above [5, 6]. These variants of weighted automata have incomparable expression power. We use the latter definition as it enables us to express long-run average and discounted sum, which are inexpressible using weighted automata over semirings. Long-run average and discounted sum are widely used in quantitative verification and define natural distances (Example 9). Moreover, unlike the semiring-based definition, the value-function-based definition extends easily from finite to infinite words.

Similarity Functions. In our work, we use similarity functions to measure the similarity between words. Let $\mathbb{Q}^\infty$ denote the set $\mathbb{Q} \cup \{\infty\}$. A similarity function $d : S \times S \to \mathbb{Q}^\infty$ is a function with the properties: $\forall x, y \in S$: (1) $d(x, y) \geq 0$ and (2) $d(x, y) = d(y, x)$. A similarity function d is also a distance (function or metric) if it satisfies the additional properties: $\forall x, y, z \in S$: (3) $d(x, y) = 0$ iff $x = y$ and (4) $d(x, z) \leq d(x, y) + d(y, z)$. We emphasize that in our work we do not need to restrict similarity functions to be distances.

An example of a similarity function is the *generalized Manhattan distance* defined as: $d_M(s,t) = \sum_{i=1}^{\infty} \mathtt{diff}(s[i], t[i])$ for infinite words s, t, where `diff` is the mismatch penalty for substituting letters. For finite words s, t, $d_M(s,t) = \sum_{i=1}^{max(|s|,|t|)} \mathtt{diff}(s[i], t[i])$. The mismatch penalty is required to be a distance function on the alphabet (extended with a special end-of-string letter # for finite words). When $\mathtt{diff}(a, b)$ is defined to be 1 for all a, b with $a \neq b$, and 0 otherwise, d_M is called the *Manhattan distance.*

Notation: We use $s_1 \otimes \ldots \otimes s_k$ to denote *convolution* of words $s_1, \ldots, s_k$, for $k > 1$. The convolution of k words merges the arguments into a single word over a k-tuple alphabet (accommodating arguments of different lengths using # letters at the ends of shorter words). Let $s_1, \ldots, s_k$ be words over alphabets $\Sigma_1, \ldots, \Sigma_k$. Let $\Sigma_1 \otimes \ldots \otimes \Sigma_k$ denote the k-tuple alphabet $(\Sigma_1 \cup \{\#\}) \times \ldots \times (\Sigma_k \cup \{\#\})$. The convolution $s_1 \otimes \ldots \otimes s_k$ is an infinite word (resp., a finite word of length $max(|s_1|, \ldots, |s_k|)$), over $\Sigma_1 \otimes \ldots \otimes \Sigma_k$, such that: for each $i \in \{1, \ldots, |s_1 \otimes \ldots \otimes s_k|\}$, $(s_1 \otimes \ldots \otimes s_k)[i] = \langle s_1[i], \ldots, s_k[i] \rangle$ (with $s_j[i] = \#$ if $i > |s_j|$). For example, the convolution $aa \otimes b \otimes add$ is the 3 letter word $\langle a, b, a\rangle\langle a, \#, d\rangle\langle \#, \#, d\rangle$.

▶ **Definition 2** (Automatic Similarity Function)**.** A similarity function $d : \Sigma_1^\omega \times \Sigma_2^\omega \mapsto \mathbb{Q}$ is called automatic if there exists a WA $\mathcal{A}_d$ over $\Sigma_1 \otimes \Sigma_2$ such that $\forall s_1 \in \Sigma_1^\omega, s_2 \in \Sigma_2^\omega$: $d(s_1, s_2) = \mathcal{L}_{\mathcal{A}_d}(s_1 \otimes s_2)$. We say that d is computed by $\mathcal{A}_d$.

One can similarly define automatic similarity functions over finite words.

3 Problem Definition

Our notion of robustness for transducers is based on the analytic notion of Lipschitz continuity. We first define K-Lipschitz robustness of functional transducers.

▶ **Definition 3** (K-Lipschitz Robustness of Functional Transducers)**.** Given a constant $K \in \mathbb{Q}$ with $K > 0$ and similarity functions $d_\Sigma : \Sigma^\omega \times \Sigma^\omega \to \mathbb{Q}^\infty$ (resp., $d_\Sigma : \Sigma^* \times \Sigma^* \to \mathbb{Q}^\infty$) and $d_\Gamma : \Gamma^\omega \times \Gamma^\omega \to \mathbb{Q}^\infty$ (resp., $d_\Gamma : \Gamma^* \times \Gamma^* \to \mathbb{Q}^\infty$), a functional transducer $\mathcal{T}$, with $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$ (resp., $[\![\mathcal{T}]\!] \subseteq \Sigma^* \times \Gamma^*$,), is called K-Lipschitz robust w. r. t. d_Σ, d_Γ if:

$$\forall s, t \in \mathrm{dom}(\mathcal{T}) : \; d_\Sigma(s,t) < \infty \Rightarrow d_\Gamma([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) \leq K d_\Sigma(s,t).$$

Recall that when $\mathcal{T}$ is an arbitrary nondeterministic transducer, for each $s \in \mathrm{dom}(\mathcal{T})$, $[\![\mathcal{T}]\!](s)$ is a set of words in Γ^ω (resp., Γ^*). Hence, we cannot use a similarity function over Γ^ω (resp., Γ^*) to define the similarity between $[\![\mathcal{T}]\!](s)$ and $[\![\mathcal{T}]\!](t)$, for $s, t \in \mathrm{dom}(\mathcal{T})$. Instead, we must use a *set-similarity function* that can compute the similarity between sets of words in Γ^ω (resp., Γ^*). We define K-Lipschitz robustness of nondeterministic transducers using such set-similarity functions (we use the notation d and D for similarity functions and set-similarity functions, respectively).

▶ **Definition 4** (K-Lipschitz Robustness of Nondeterministic Transducers)**.** Given a constant $K \in \mathbb{Q}$ with $K > 0$, a similarity function $d_\Sigma : \Sigma^\omega \times \Sigma^\omega \to \mathbb{Q}^\infty$ (resp., $d_\Sigma : \Sigma^* \times \Sigma^* \to \mathbb{Q}^\infty$) and a set-similarity function $D_\Gamma : 2^{\Gamma^\omega} \times 2^{\Gamma^\omega} \to \mathbb{Q}^\infty$ (resp., $D_\Gamma : 2^{\Gamma^*} \times 2^{\Gamma^*} \to \mathbb{Q}^\infty$), a nondeterministic transducer $\mathcal{T}$, with $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$ (resp. $[\![\mathcal{T}]\!] \subseteq \Sigma^* \times \Gamma^*$), is called K-Lipschitz robust w. r. t. d_Σ, D_Γ if:

$$\forall s, t \in \mathrm{dom}(\mathcal{T}):\ d_\Sigma(s,t) < \infty \Rightarrow D_\Gamma([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) \leq K d_\Sigma(s,t).$$

In what follows, we use K-robustness to denote K-Lipschitz robustness. The results in the remainder of this paper hold both for machines processing ω-words as well as for those processing finite words. To keep the presentation clean, we present all results in the context of machines over ω-words, making a distinction as needed. Moreover, we only present some (partial) proofs in this paper. We direct the interested reader to [15] for the complete proofs.

4 Synchronized (Functional) Transducers

In this section, we define a class of functional transducers which admits a decision procedure for K-robustness.

▶ **Definition 5** (Synchronized Transducers). A functional transducer $\mathcal{T}$ with $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$ is synchronized iff there exists an automaton $\mathcal{A}_\mathcal{T}$ over $\Sigma \otimes \Gamma$ recognizing the language $\{s \otimes [\![\mathcal{T}]\!](s) : s \in \mathrm{dom}(\mathcal{T})\}$.

Let $\mathcal{T}$ be an arbitrary functional transducer. In each transition, $\mathcal{T}$ reads a single input letter and may generate an empty output word or an output word longer than a single letter. To process such non-aligned input and output words, the automaton $\mathcal{A}_\mathcal{T}$ needs to internally implement a buffer. Thus, $\mathcal{T}$ is synchronized iff there is a bound B on the required size of such a buffer. We can use this observation to check if $\mathcal{T}$ is synchronized. Note that letter-to-letter transducers are synchronized, with B being 0.

▶ **Proposition 6.** *Synchronicity of a functional transducer is decidable in polynomial time.*

Synchronized transducers admit an automata-theoretic decision procedure for checking K-robustness w. r. t. similarity functions satisfying certain properties.

▶ **Theorem 7.** *For every $f \in$ ValFunc, if d_Σ, d_Γ are similarity functions computed by functional f-WA $\mathcal{A}_{d_\Sigma}$, $\mathcal{A}_{d_\Gamma}$, respectively, and $\mathcal{T}$ is a synchronized transducer, K-robustness of $\mathcal{T}$ w. r. t. d_Σ, d_Γ is decidable in polynomial time in the sizes of $\mathcal{T}$, $\mathcal{A}_{d_\Sigma}$ and $\mathcal{A}_{d_\Gamma}$.*

We show that for every $f \in$ ValFunc, if the conditions of Theorem 7 are met, K-robustness of $\mathcal{T}$ can be reduced to the emptiness problem for f-weighted automata, which is decidable in polynomial time.

Similarity functions computed by nondeterministic automata. If we permit the weighted automata computing the similarity functions d_Σ, d_Γ to be nondeterministic, K-robustness becomes undecidable. We can show that the universality problem for nondeterministic weighted automata reduces to checking 1-robustness. Indeed, given a nondeterministic weighted automaton $\mathcal{A}$, consider (1) d_Σ such that $\forall s, t \in \Sigma^\omega$: $d_\Sigma(s,t) = \lambda$ if $s = t$, and undefined otherwise, (2) $\mathcal{T}$ encoding the identity function, and (3) d_Γ such that $\forall s', t' \in \Sigma^\omega$: $d_\Gamma(s', t') = \mathcal{L}_\mathcal{A}(s')$ if $s' = t'$, and undefined otherwise. Note that d_Γ is computed by a nondeterministic weighted automaton obtained from $\mathcal{A}$ by changing each transition (q, a, q') in $\mathcal{A}$ to $(q, (a, a), q')$ while preserving the weight. Then, $\mathcal{T}$ is 1-robust w. r. t. d_Σ, d_Γ iff for all words s, $\mathcal{L}_\mathcal{A}(s) \leq \lambda$. Since the universality problem for f-weighted automata is undecidable (e.g., for $f =$ Sum), it follows that checking 1-robustness of transducers with similarity functions computed by nondeterministic weighted automata is undecidable.

We now present examples of synchronized transducers and automatic similarity functions satisfying the conditions of Theorem 7.

▶ **Example 8** (Mealy machines and generalized Manhattan distances.). Mealy machines are perhaps the most widely used transducer model. Prior work [22] has shown decidability of robustness of Mealy machines with respect to generalized Manhattan distances given a fixed bound on the amount of input perturbation. In what follows, we argue the decidability of robustness of Mealy machines (processing infinite words) with respect to generalized Manhattan distances in the presence of unbounded input perturbation.

A Mealy machine $\mathcal{T} : (\Sigma, \Gamma, Q, \{q_0\}, E_{\mathcal{T}}, Q)$ is a synchronized transducer with $\mathcal{A}_{\mathcal{T}}$ given by $(\Sigma \otimes \Gamma, Q, \{q_0\}, E_{\mathcal{A}_{\mathcal{T}}}, Q)$, where $E_{\mathcal{A}_{\mathcal{T}}} = \{(q, a \otimes a', q') : (q, a, a', q') \in E_{\mathcal{T}}\}$. The generalized Manhattan distance $d_M : \Sigma^\omega \times \Sigma^\omega \to \mathbb{Q}^\infty$ can be computed by a functional Sum-weighted automaton $\mathcal{A}_M$ given by the tuple $(\Sigma \otimes \Sigma, \{q_0\}, \{q_0\}, E_M, \{q_0\}, c)$. Here, q_0 is the initial as well as the accepting state, $E_M = \{(q_0, a \otimes b, q_0) : a \otimes b \in \Sigma \otimes \Sigma\}$, and the weight of each transition $(q_0, a \otimes b, q_0)$ equals $\texttt{diff}(a, b)$.

Thus, all the conditions of Theorem 7 are satisfied. K-robustness of Mealy machines, with d_Σ, d_Γ being the generalized Manhattan distance, is decidable in polynomial time.

▶ **Example 9** (Piecewise-linear functions.). Let us use $\underline{q}$ to denote an infinite word over $\{0, \dots, 9, +, -\}$ representing the fractional part of a real number in base 10. E. g., $\underline{-0.21} = -21$ and $\underline{\pi - 3} = 1415\dots$ Then, $\underline{q_1} \otimes \dots \otimes \underline{q_k}$ is a word over $\{0, \dots, 9, +, -\} \otimes \dots \otimes \{0, \dots, 9, +, -\}$ that represents a k-tuple of real numbers $q_1, \dots, q_k$ from the interval $(-1, 1)$. Now, observe that one can define letter-to-letter transducers that compute the following functions: (1) swapping of arguments, $[\![\mathcal{T}]\!](\underline{q_1}, \dots, \underline{q_l}, \dots, \underline{q_m}, \dots, \underline{q_k}) = (\underline{q_1}, \dots, \underline{q_m}, \dots, \underline{q_l}, \dots, \underline{q_k})$, (2) addition, $[\![\mathcal{T}]\!](\underline{q_1}, \dots, \underline{q_k}) = (\underline{q_1 + q_2}, \underline{q_2}, \dots, \underline{q_k})$, (3) multiplication by a constant c, $[\![\mathcal{T}]\!](\underline{q_1}, \dots, \underline{q_k}) = (\underline{cq_1}, \dots, \underline{cq_k})$, (4) projection, $[\![\mathcal{T}]\!](\underline{q_1}, \dots, \underline{q_k}) = (\underline{q_1}, \dots, \underline{q_{k-1}})$, and (5) conditional expression, $[\![\mathcal{T}]\!](\underline{q_1}, \dots, \underline{q_k})$ equals $[\![\mathcal{T}_1]\!](\underline{q_1}, \dots, \underline{q_k})$, if $\underline{q_1} > 0$, and $[\![\mathcal{T}_2]\!](\underline{q_1}, \dots, \underline{q_k})$ otherwise. We assume that the transducers reject if the results of the corresponding functions lie outside the interval $(-1, 1)$. We can model a large class of piecewise-linear functions using transducers obtained by composition of transducers (1)-(5). The resulting transducers are functional letter-to-letter transducers.

Now, consider d_Σ, d_Γ defined as the *L1*-norm over $\mathbb{R}^k$, i.e., $d_\Sigma(\underline{q_1} \otimes \dots \otimes \underline{q_k}, \underline{q'_1} \otimes \dots \otimes \underline{q'_k})$ $= d_\Gamma(\underline{q_1} \otimes \dots \otimes \underline{q_k}, \underline{q'_1} \otimes \dots \otimes \underline{q'_k}) = \sum_{i=1}^{k} \text{abs}(q_i - q'_i)$. Observe that d_Σ, d_Γ can be computed by deterministic Disc$_\delta$-weighted automata, with $\delta = \frac{1}{10}$. Therefore, 1-robustness of $\mathcal{T}$ can be decided in polynomial time (Theorem 7). Finally, note that K-robustness of a transducer computing a piecewise-linear function h w. r. t. the above similarity functions is equivalent to Lipschitz continuity of h with coefficient K.

5 Functional Transducers

It was shown in [22] that checking K-robustness of a functional transducer w. r. t. to a fixed bound on the amount of input perturbation is decidable. In what follows, we show that when the amount of input perturbation is unbounded, the robustness problem becomes undecidable even for deterministic transducers.

▶ **Theorem 10.** *1-robustness of deterministic transducers is undecidable.*

Proof. The Post Correspondence Problem (PCP) is defined as follows. Given a set of word pairs $\{\langle v_1, w_1\rangle, \dots, \langle v_k, w_k\rangle\}$, does there exist a sequence of indices $i_1, \dots, i_n$ such that $v_{i_1} \cdot \dots \cdot v_{i_n} = w_{i_1} \cdot \dots \cdot w_{i_n}$? PCP is known to be undecidable.

Let $\mathcal{G}_{pre} = \{\langle v_1, w_1\rangle, \dots, \langle v_k, w_k\rangle\}$ be a PCP instance with $v_i, w_i \in \{a, b\}^*$ for each $i \in [1, k]$. We define a new instance $\mathcal{G} = \mathcal{G}_{pre} \cup \{\langle v_{k+1}, w_{k+1}\rangle\}$, where $\langle v_{k+1}, w_{k+1}\rangle = \langle \$, \$\rangle$. Observe that for $i_1, \dots, i_n \in [1, k]$, $i_1, \dots, i_n, k+1$ is a solution of $\mathcal{G}$ iff $i_1, \dots, i_n$ is a solution of $\mathcal{G}_{pre}$. We define a deterministic transducer $\mathcal{T}$ processing finite words and generalized Manhattan distances d_Σ, d_Γ such that $\mathcal{T}$ is *not* 1-robust w. r. t. d_Σ, d_Γ iff $\mathcal{G}$ has a solution of the form $i_1, \dots, i_n, k+1$, with $i_1, \dots, i_n \in [1, k]$.

We first define $\mathcal{T}$, which translates indices into corresponding words from the PCP instance $\mathcal{G}$. The input alphabet Σ is the set of indices from $\mathcal{G}$, marked with a *polarity*, L or R, denoting whether an index i, corresponding to a pair $\langle v_i, w_i\rangle \in \mathcal{G}$, is translated to v_i or w_i. Thus, $\Sigma = \{1, \dots, k+1\} \times \{L, R\}$. The output alphabet Γ is the alphabet of words in $\mathcal{G}$, marked with a polarity. Thus, $\Gamma = \{a, b, \$\} \times \{L, R\}$. The domain of $[\![\mathcal{T}]\!]$ is described by the following regular expression: $\mathrm{dom}(\mathcal{T}) = \Sigma_L^* \langle k+1, L\rangle + \Sigma_R^* \langle k+1, R\rangle$, where for $P \in \{L, R\}$, $\Sigma_P = \{1, \dots, k\} \times \{P\}$. Thus, $\mathcal{T}$ only processes input words over letters with the same polarity, rejecting upon reading an input letter with a polarity different from that of the first input letter. Moreover, $\mathcal{T}$ accepts iff the first occurrence of $\langle k+1, L\rangle$ or $\langle k+1, R\rangle$ is in the last position of the input word. Note that the domain of $\mathcal{T}$ is *prefix-free*, i.e., if $s, t \in \mathrm{dom}(\mathcal{T})$ and s is a prefix of t, then $s = t$. Let u^P denote the word $u \otimes P^{|u|}$. Along accepting runs, $\mathcal{T}$ translates each input letter $\langle i, L\rangle$ to v_i^L and each letter $\langle i, R\rangle$ to w_i^R, where $\langle v_i, w_i\rangle$ is the i^{th} word pair of $\mathcal{G}$. Thus, the function computed by $\mathcal{T}$ is:

$$[\![\mathcal{T}]\!](\langle i_1, L\rangle \dots \langle i_n, L\rangle \langle k+1, L\rangle) = v_{i_1}^L \dots v_{i_n}^L v_{k+1}^L$$
$$[\![\mathcal{T}]\!](\langle i_1, R\rangle \dots \langle i_n, R\rangle \langle k+1, R\rangle) = w_{i_1}^R \dots w_{i_n}^R w_{k+1}^R$$

We define the output similarity function d_Γ as a generalized Manhattan distance with the following *symmetric* $\mathtt{diff}_\Gamma$ where $P, Q \in \{L, R\}$ and $\alpha, \beta \in \{a, b, \$\}$ with $\alpha \neq \beta$:

$$\mathtt{diff}_\Gamma(\langle \alpha, P\rangle, \langle \alpha, P\rangle) = 0 \qquad \mathtt{diff}_\Gamma(\langle \alpha, L\rangle, \langle \alpha, R\rangle) = 2$$
$$\mathtt{diff}_\Gamma(\langle \alpha, P\rangle, \langle \beta, Q\rangle) = 1 \qquad \mathtt{diff}_\Gamma(\langle \alpha, P\rangle, \#) = 1$$

Note that for $s', t' \in \Gamma^*$ with different polarities, $d_\Gamma(s', t')$ equals the sum of $max(|s'|, |t'|)$ and $\mathcal{N}(s', t')$, where $\mathcal{N}(s', t')$ is the number of positions in which s' and t' agree on the first components of their letters.

We define a projection π as $\pi(\langle i_1, P_1\rangle \langle i_2, P_2\rangle \dots \langle i_n, P_n\rangle) = i_1 i_2 \dots i_n$, where $i_1, \dots, i_n \in [1, k+1]$ and $P_1, \dots, P_n \in \{L, R\}$. We define the input similarity function d_Σ as a generalized Manhattan distance such that $d_\Sigma(s, t)$ is finite iff $\pi(s)$ is a prefix of $\pi(t)$ or vice versa. We define d_Σ using the following *symmetric* $\mathtt{diff}_\Sigma$ where $P, Q \in \{L, R\}$ and $i, j \in [1, k+1]$ with $i \neq j$:

$$\mathtt{diff}_\Sigma(\langle i, P\rangle, \langle i, P\rangle) = 0 \qquad \mathtt{diff}_\Sigma(\langle i, P\rangle, \langle j, Q\rangle) = \infty$$
$$\mathtt{diff}_\Sigma(\langle i, L\rangle, \langle i, R\rangle) = |v_i| + |w_i|, \text{ if } i \in [1, k] \qquad \mathtt{diff}_\Sigma(\langle i, P\rangle, \#) = \infty$$
$$\mathtt{diff}_\Sigma(\langle k+1, L\rangle, \langle k+1, R\rangle) = 1$$

Thus, for all $s, t \in \mathrm{dom}(\mathcal{T})$, $d_\Sigma(s, t) < \infty$ iff one of the following holds:

(i) for some $P \in \{L, R\}$, $s = t = \langle i_1, P\rangle \dots \langle i_n, P\rangle \langle k+1, P\rangle$, or,

(ii) $s = \langle i_1, L\rangle \dots \langle i_n, L\rangle \langle k+1, L\rangle$ and $t = \langle i_1, R\rangle \dots \langle i_n, R\rangle \langle k+1, R\rangle$.

In case (i), $d_\Sigma(s, t) = d_\Gamma([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) = 0$. In case (ii), $d_\Sigma(s, t) = |[\![\mathcal{T}]\!](s)| + |[\![\mathcal{T}]\!](t)| - 1$ and $d_\Gamma([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) = max(|[\![\mathcal{T}]\!](s)|, |[\![\mathcal{T}]\!](t)|) + \mathcal{N}([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t))$. Thus, $d_\Gamma([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) > d_\Sigma(s, t)$ iff $\mathcal{N}([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) = min(|[\![\mathcal{T}]\!](s)|, |[\![\mathcal{T}]\!](t)|)$. Since the letters $\langle \$, L\rangle, \langle \$, R\rangle$ occur exactly once in $[\![\mathcal{T}]\!](s)$, $[\![\mathcal{T}]\!](t)$, respectively, at the end of each word, $\mathcal{N}([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) = min(|[\![\mathcal{T}]\!](s)|, |[\![\mathcal{T}]\!](t)|)$ iff $|[\![\mathcal{T}]\!](s)| = |[\![\mathcal{T}]\!](t)|$ and $\pi([\![\mathcal{T}]\!](s)) = \pi([\![\mathcal{T}]\!](t))$, which holds iff $\mathcal{G}$ has a solution. Therefore, $\mathcal{T}$ is *not* 1-robust w. r. t. d_Σ, d_Γ iff $\mathcal{G}$ has a solution. ◀

We have shown that checking 1-robustness w. r. t. generalized Manhattan distances is undecidable. Observe that for every $K > 0$, K-robustness can be reduced to 1-robustness by scaling the output distance by K. We conclude that checking K-robustness is undecidable for any fixed K. In contrast, if K is not fixed, checking if there exists K such that $\mathcal{T}$ is K-robust w. r. t. d_Σ, d_Γ is decidable for transducers processing finite words.

Let us define a functional transducer $\mathcal{T}$ to be *robust* w. r. t. d_Σ, d_Γ if there exists K such that $\mathcal{T}$ is K-robust w. r. t. d_Σ, d_Γ.

▶ **Proposition 11.** *Let $\mathcal{T}$ be a given functional transducer processing finite words and d_Σ, d_Γ be instances of the generalized Manhattan distance.*

1. *Robustness of $\mathcal{T}$ is decidable in* CO-NP*.*
2. *One can compute $K_{\mathcal{T}}$ such that $\mathcal{T}$ is robust iff $\mathcal{T}$ is $K_{\mathcal{T}}$-robust.*

Proof sketch. Given $\mathcal{T}$, one can easily construct a *trim*[2] functional transducer $\mathcal{P}_{\mathcal{T}}$ such that $[\![\mathcal{P}_{\mathcal{T}}]\!](s,t) = (s',t')$ iff $[\![\mathcal{T}]\!](s) = s'$ and $[\![\mathcal{T}]\!](t) = t'$. We show that $\mathcal{T}$ is not robust w. r. t. generalized Manhattan distances iff there exists some cycle in $\mathcal{P}_{\mathcal{T}}$ satisfying certain properties. Checking the existence of such a cycle is in NP. If such a cycle exists, one can construct paths in $\mathcal{P}_{\mathcal{T}}$ through the cycle, labeled with input words (s,t) and output words (s',t'), with $d_\Gamma(s',t') > K d_\Sigma(s,t)$ for *any* K. Conversely, if there exists no such cycle, one can compute $K_{\mathcal{T}}$ such that $\mathcal{T}$ is $K_{\mathcal{T}}$-robust. It follows that one can compute $K_{\mathcal{T}}$ such that $\mathcal{T}$ is robust iff $\mathcal{T}$ is $K_{\mathcal{T}}$-robust. ◀

5.1 Beyond Synchronized Transducers

In this section, we present an approach for natural extensions of Theorem 7.

Isometry approach. We say that a transducer $\mathcal{T}$ is a (d_Λ, d_Δ)-*isometry* if and only if for all $s,t \in \mathrm{dom}(\mathcal{T})$ we have $d_\Lambda(s,t) = d_\Delta([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t))$.

▶ **Proposition 12.** *Let $\mathcal{T}, \mathcal{T}'$ be functional transducers with $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$ and $[\![\mathcal{T}']\!] \subseteq \Lambda^\omega \times \Delta^\omega$. Assume that there exist transducers $\mathcal{T}^I$ and $\mathcal{T}^O$ such that $\mathcal{T}^I$ is a (d_Σ, d_Λ)-isometry, $\mathcal{T}^O$ is a (d_Δ, d_Γ)-isometry and $[\![\mathcal{T}]\!] = [\![\mathcal{T}^O \circ (\mathcal{T}' \circ \mathcal{T}^I)]\!]$. Then, for every $K > 0$, $\mathcal{T}$ is K-robust w. r. t. d_Σ, d_Γ if and only if $\mathcal{T}'$ is K-robust w. r. t. d_Λ, d_Δ.*

▶ **Example 13** (Stuttering)**.** For a given word w we define the *stuttering pruned* word $\textsc{Stutter}(w)$ as the result of removing from w letters that are the same as the previous letter. E. g. $\textsc{Stutter}(\mathbf{ba}aa\mathbf{c}c\mathbf{a}aa\mathbf{b}) = bacab$.

Consider a transducer $\mathcal{T}$ and a similarity function d_Σ over finite words that are *stuttering invariant*, i.e., for all $s,t \in \mathrm{dom}(\mathcal{T})$, if $\textsc{Stutter}(s) = \textsc{Stutter}(t)$, then $[\![\mathcal{T}]\!](s) = [\![\mathcal{T}]\!](t)$ and for every $u \in \Sigma^*$, $d_\Sigma(s,u) = d_\Sigma(t,u)$. In addition, we assume that for every $s \in \mathrm{dom}(\mathcal{T})$, $|[\![\mathcal{T}]\!](s)| = |\textsc{Stutter}(s)|$.

Observe that these assumptions imply that: (1) the projection transducer $\mathcal{T}^\pi$ defined such that $[\![s]\!] = \textsc{Stutter}(s)$ is a (d_Σ, d_Σ)-isometry, (2) the transducer $\mathcal{T}^S$ obtained by restricting the domain of $\mathcal{T}$ to *stuttering-free* words, i.e., the set $\{w \in \mathrm{dom}(\mathcal{T}) : \textsc{Stutter}(w) = w\}$, is a synchronized transducer[3], and (3) $[\![\mathcal{T}]\!] = [\![\mathcal{T}^I \circ (\mathcal{T}^S \circ \mathcal{T}^\pi)]\!]$, where $\mathcal{T}^I$ defines the identity function over Γ^*. Therefore, by Proposition 12, in order to check K-robustness of $\mathcal{T}$, it suffices to check K-robustness of $\mathcal{T}^S$. Since $\mathcal{T}^S$ is a synchronized transducer, K-robustness of

[2] $\mathcal{P}_{\mathcal{T}}$ is trim if every state in $\mathcal{P}_{\mathcal{T}}$ is reachable from the initial state and some final state is reachable from every state in $\mathcal{P}_{\mathcal{T}}$.

[3] Note that any functional transducer $\mathcal{T}$ with the property: for every $s \in \mathrm{dom}(\mathcal{T})$, $|[\![\mathcal{T}]\!](s)| = |s|$, is a synchronized transducer.

$\mathcal{T}^S$ can be effectively checked, provided the similarity functions d_Σ, d_Γ satisfy the conditions of Theorem 7.

▶ **Example 14** (Letter-to-multiple-letters transducers). Consider a transducer $\mathcal{T}$ which on every transition outputs a 2-letter word[4]. Although, $\mathcal{T}$ is not synchronized, it can be transformed to a letter-to-letter transducer $\mathcal{T}^D$, whose output alphabet is $\Gamma \times \Gamma$. The transducer $\mathcal{T}^D$ is obtained from $\mathcal{T}$ by substituting each output word ab to a single letter $\langle a, b\rangle$ from $\Gamma \times \Gamma$. We can use $\mathcal{T}^D$ to decide K-robustness of $\mathcal{T}$ in the following way. First, we define transducers $\mathcal{T}^I, \mathcal{T}^{\text{pair}}$ such that $\mathcal{T}^I$ computes the identity function over Σ^ω and $\mathcal{T}^{\text{pair}}$ is a transducer representing the function $[\![\mathcal{T}^{\text{pair}}]\!](\langle a_1, b_1\rangle\langle a_2, b_2\rangle \ldots) = a_1 b_1 a_2 b_2 \ldots$. Observe that $[\![\mathcal{T}]\!] = [\![\mathcal{T}^{\text{pair}} \circ (\mathcal{T}^D \circ \mathcal{T}^I)]\!]$. Second, we define d_Γ^D as follows: $\forall s, t \in (\Sigma \times \Sigma)^\omega$, $d_\Gamma^D(s, t) = d_\Gamma([\![\mathcal{T}^{\text{pair}}]\!](s), [\![\mathcal{T}^{\text{pair}}]\!](t))$. Observe that $\mathcal{T}^I$ is a (d_Σ, d_Σ)-isometry and $\mathcal{T}^{\text{pair}}$ is a (d_Γ^D, d_Γ)-isometry. Thus, K-robustness of $\mathcal{T}$ w. r. t. d_Σ, d_Γ reduces to K-robustness of the letter-to-letter transducer $\mathcal{T}^D$ w. r. t. d_Σ, d_Γ^D, which can be effectively checked (Theorem 7).

6 Nondeterministic Transducers

Let $\mathcal{T}$ be a nondeterministic transducer with $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$. Let d_Σ be an automatic similarity function for computing the similarity between input words in Σ^*. As explained in Sec. 3, the definition of K-robust nondeterministic transducers involves set-similarity functions that can compute the similarity between sets of output words in Γ^ω. In this section, we examine the K-robustness problem of $\mathcal{T}$ w. r. t. d_Σ and three classes of such set-similarity functions.

Let d_Γ be an automatic similarity function for computing the similarity between output words in Γ^ω. We first define three set-similarity functions induced by d_Γ.

▶ **Definition 15.** Given sets A, B of words in Γ^ω, we consider the following set-similarity functions induced by d_Γ:

(i) Hausdorff set-similarity function $D_\Gamma^H(A, B)$ induced by d_Γ:

$$D_\Gamma^H(A, B) = max\{\, \sup_{s\in A} \inf_{t\in B} d_\Gamma(s, t), \sup_{s\in B} \inf_{t\in A} d_\Gamma(s, t)\,\}$$

(ii) Inf-inf set-similarity function $D_\Gamma^{\inf}(A, B)$ induced by d_Γ:

$$D_\Gamma^{\inf}(A, B) = \inf_{s\in A} \inf_{t\in B} d_\Gamma(s, t)$$

(iii) Sup-sup set-similarity function $D_\Gamma^{\sup}(A, B)$ induced by d_Γ:

$$D_\Gamma^{\sup}(A, B) = \sup_{s\in A} \sup_{t\in B} d_\Gamma(s, t)$$

Of the above set-similarity functions, only the Hausdorff set-similarity function is a distance function (if d_Γ is a distance function).

Note that when $\mathcal{T}$ is a functional transducer, each set-similarity function above reduces to d_Γ. Hence, K-robustness of a functional transducer $\mathcal{T}$ w. r. t. d_Σ, D_Γ and K-robustness of $\mathcal{T}$ w. r. t. d_Σ, d_Γ coincide. As K-robustness of functional transducers in undecidable (Theorem 10), K-robustness of nondeterministic transducers w. r. t. the above set-similarity functions is undecidable as well.

Recall from Theorem 7 that K-robustness of a synchronized (functional) transducer is decidable w. r. t. certain automatic similarity functions. In particular, K-robustness of

[4] One can easily generalize this example to any fixed number.

Mealy machines is decidable when d_Σ, d_Γ are generalized Manhattan distances. In contrast, K-robustness of nondeterministic letter-to-letter transducers is undecidable w. r. t. the Hausdorff and Inf-inf set-similarity functions even when d_Σ, d_Γ are generalized Manhattan distances. Among the above defined set-similarity functions, K-robustness of nondeterministic transducers is decidable only w. r. t. the Sup-sup set-similarity function.

▶ **Theorem 16.** *Let d_Σ, d_Γ be computed by functional weighted-automata. Checking K-robustness of nondeterministic letter-to-letter transducers w. r. t. d_Σ, D_Γ induced by d_Γ is*

(i) *undecidable if D_Γ is the Hausdorff set-similarity function,*
(ii) *undecidable if D_Γ is the Inf-inf set-similarity function, and*
(iii) *decidable if D_Γ is the Sup-sup set-similarity function and d_Σ, d_Γ satisfy the conditions of Theorem 7.*

Proof of (iii). We can encode nondeterministic choices of $\mathcal{T}$, with $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$, in an extended input alphabet $\Sigma \times \Lambda$. We construct a deterministic transducer $\mathcal{T}^e$ such that for every $s \in \Sigma^\omega$, $\{[\![\mathcal{T}^e]\!](\langle s, \lambda\rangle) : \langle s, \lambda\rangle \in \mathrm{dom}(\mathcal{T}^e)\} = [\![\mathcal{T}]\!](s)$. We also define d^e_Σ such that for all $\langle s, \lambda_1\rangle, \langle t, \lambda_2\rangle \in (\Sigma \times \Lambda)^\omega$, $d^e_\Sigma(\langle s, \lambda_1\rangle, \langle t, \lambda_2\rangle) = d_\Sigma(s,t)$. Then, $\mathcal{T}$ is K-robust w. r. t. $d_\Sigma, D^{\sup}_\Gamma$ induced by d_Γ iff $\mathcal{T}^e$ is K-robust w. r. t. d^e_Σ, d_Γ. Indeed, a nondeterministic transducer $\mathcal{T}$ is K-robust w. r. t. d_Σ, $D^{\sup}_\Gamma$ induced by d_Γ iff for all input words $s,t \in \mathrm{dom}(\mathcal{T})$ and for all outputs $s' \in [\![\mathcal{T}]\!](s), t' \in [\![\mathcal{T}]\!](t)$, $d_\Sigma(s,t) < \infty$ implies $d_\Gamma(s',t') \leq K d_\Sigma(s,t)$. ◀

7 Related Work

In early work [19], [7, 8] on continuity and robustness analysis, the focus is on software programs manipulating numbers. In [19], the authors compute the maximum deviation of a program's output given the maximum possible perturbation in a program input. In [7], the authors formalize $\epsilon - \delta$ continuity of programs and present sound proof rules to prove continuity of programs. In [8], the authors formalize robustness of programs as Lipschitz continuity and present a sound program analysis for robustness verification. While arrays of numbers are considered in [8], the size of an array is immutable.

More recent papers have aimed to develop a notion of robustness for reactive systems. In [23], the authors present polynomial-time algorithms for the analysis and synthesis of robust transducers. Their notion of robustness is one of input-output stability, that bounds the output deviation from disturbance-free behaviour under bounded disturbance, as well as the persistence of the effect of a sporadic disturbance. Their distances are measured using cost functions that map *each* string to a nonnegative integer. In [18, 4, 2], the authors develop different notions of robustness for reactive systems, with ω-regular specifications, interacting with uncertain environments. In [9], the authors present a polynomial-time algorithm to decide robustness of sequential circuits modeled as Mealy machines, w. r. t. a *common suffix distance* metric. Their notion of robustness also bounds the persistence of the effect of a sporadic disturbance.

Recent work in [21] and [22] formalized and studied robustness of systems modeled using transducers, in the presence of bounded perturbation. The work in [21] focussed on the outputs of synchronous networks of Mealy machines in the presence of channel perturbation. The work in [22] focussed on the outputs of functional transducers in the presence of input perturbation. Both papers presented decision procedures for robustness verification w. r. t. specific distance functions such as Manhattan and Levenshtein distances.

8 Conclusion

In this paper, we studied the K-Lipschitz robustness problem for finite-state transducers. While the general problem is undecidable, we identified decidability criteria that enable reduction of K-robustness to the emptiness problem for weighted automata.

In the future, we wish to extend our work in two directions. We plan to study robustness of other computational models. We also wish to investigate synthesis of robust transducers.

References

1 S. Almagor, U. Boker, and O. Kupferman. What's Decidable about Weighted Automata? In *ATVA*, pages 482–491. LNCS 6996, Springer, 2011.

2 R. Bloem, K. Greimel, T. Henzinger, and B. Jobstmann. Synthesizing Robust Systems. In *Formal Methods in Computer Aided Design (FMCAD)*, pages 85–92, 2009.

3 R. K. Bradley and I. Holmes. Transducers: An Emerging Probabilistic Framework for Modeling Indels on Trees. *Bioinformatics*, 23(23):3258–3262, 2007.

4 P. Cerny, T. Henzinger, and A. Radhakrishna. Simulation Distances. In *Conference on Concurrency Theory (CONCUR)*, pages 253–268, 2010.

5 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *FCT*, volume 5699 of *LNCS*, pages 3–13. Springer, 2009.

6 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.

7 S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity Analysis of Programs. In *Principles of Programming Languages (POPL)*, pages 57–70, 2010.

8 S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving Programs Robust. In *Foundations of Software Engineering (FSE)*, pages 102–112, 2011.

9 L. Doyen, T. A. Henzinger, A. Legay, and D. Ničković. Robustness of Sequential Circuits. In *Application of Concurrency to System Design (ACSD)*, pages 77–84, 2010.

10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.

11 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag New York, Inc., New York, USA, 1996.

12 E. M. Gurari and O. H. Ibarra. A Note on Finitely-Valued and Finitely Ambiguous Transducers. *Mathematical Systems Theory*, 16(1):61–66, 1983.

13 D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

14 T. A. Henzinger. Two Challenges in Embedded Systems Design: Predictability and Robustness. *Philosophical Transactions of the Royal Society*, 366:3727–3736, 2008.

15 T. A. Henzinger, J. Otop, and R. Samanta. Lipschitz Robustness of Finite-state Transducers. *CoRR*, abs/1404.6452, 2014.

16 K. Zhou and J. C. Doyle and K. Glover. *Robust and Optimal Control*. Prentice Hall, 1996.

17 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *IJAC*, 4(3):405–426, 1994.

18 R. Majumdar, E. Render, and P. Tabuada. A Theory of Robust Omega-regular Software Synthesis. *ACM Transactions on Embedded Computing Systems*, 13, 2013.

19 R. Majumdar and I. Saha. Symbolic Robustness Analysis. In *IEEE Real-Time Systems Symposium*, pages 355–363, 2009.

20 M. Mohri. Finite-state Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, 1997.

21 R. Samanta, J. V. Deshmukh, and S. Chaudhuri. Robustness Analysis of Networked Systems. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 229–247, 2013.

22 R. Samanta, J. V. Deshmukh, and S. Chaudhuri. Robustness Analysis of String Transducers. In *ATVA*, pages 427–441. LNCS 8172, Springer, 2013.

23 P. Tabuada, A. Balkan, S. Y. Caliskan, Y. Shoukry, and R. Majumdar. Input-Output Robustness for Discrete Systems. In *International Conference on Embedded Software (EMSOFT)*, 2012.

24 M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjørner. Symbolic Finite State Transducers: Algorithms and Applications. In *Principles of Programming Languages (POPL)*, pages 137–150, 2012.

Separating Cook Completeness from Karp-Levin Completeness Under a Worst-Case Hardness Hypothesis[*]

Debasis Mandal, A. Pavan, and Rajeswari Venugopalan

Department of Computer Science, Iowa State University, USA
`{debasis,pavan,dvaithi}@iastate.edu`

Abstract

We show that there is a language that is Turing complete for NP but not many-one complete for NP, under a *worst-case* hardness hypothesis. Our hypothesis asserts the existence of a non-deterministic, double-exponential time machine that runs in time $O(2^{2^{n^c}})$ (for some $c > 1$) accepting Σ^* whose accepting computations cannot be computed by bounded-error, probabilistic machines running in time $O(2^{2^{\beta 2^{n^c}}})$ (for some $\beta > 0$). This is the first result that separates completeness notions for NP under a worst-case hardness hypothesis.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Cook reduction, Karp reduction, NP-completeness, Turing completeness, many-one completeness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.445

1 Introduction

The notion of polynomial-time reductions is pervasive in theoretical computer science. In addition to their critical role in defining NP-completeness, polynomial-time reductions play an important role in establishing several results in various areas such as complexity theory, cryptography, learning theory etc. Informally, reductions translate instances of one problem to instances of another problem; a problem A is polynomial-time reducible to a problem B if A can be solved in polynomial-time by making queries to problem B. By varying the manner in which the queries are allowed to make, we obtain a wide spectrum of reductions. At one end of the spectrum is *Cook/Turing reduction* where multiple queries are allowed and the ith query made depends on answers to previous queries. On the other end is the most restrictive reduction, *Karp-Levin/many-one reduction*, where each positive instance of problem A is mapped to a positive instance of problem B, and so are the negative instances. In between are *truth-table/non-adaptive reductions*, and *bounded truth-table reductions*. Interestingly, the seminal paper of Cook [7] used Turing reduction to define NP-completeness, whereas the works of Karp [16] and Levin [19] used many-one reductions.

Understanding the differences between many-one reductions and Turing reductions is one of the fundamental problems in complexity theory. Compared to many-one reductions, our knowledge about Turing reductions is limited. Extending certain assertions that are known to be true for many-one reductions to the case of Turing reductions yield much sought after separation results in complexity theory. For example, it is known that polynomial time many-one complete sets for EXP are not sparse [24]. Extending this result to the case of

* Research Supported in part by NSF grants CCF: 0916797 and CCF: 1421163.

Turing reductions implies that EXP does not have polynomial-size circuits. In the context of resource-bounded measure, it is known that “small span theorem” holds for many-one reductions. Establishing a similar result for Turing reductions separates EXP from BPP [15]. In addition, Turing reductions are crucial to define the Polynomial-time hierarchy.

The differences between various types of polynomial-time reductions have been studied in different contexts. Selman [23] showed that if NE $\cap$ co-NE does not equal E, then there exist languages A and B in NP such that A polynomial-time Turing reduces to B, but does not polynomial-time many-one reduce to B. Aida *et al.* [1] showed a similar result in the average-case world; if P does not equal NP, then there is a distributional problems (A, μ_A) and (B, μ_B) in DistNP such that (A, μ_A) Turing reduces to (B, μ_B) but does not many-one reduce to (B, μ_B). The differences between Turing and truth-table reductions have been studied extensively in the context of random self-reductions and coherence [4, 8, 9, 11]. For example, Feigenbaum *et al.* [8] showed that if nondeterministic triple exponential time is not in bounded-error, probabilistic triple exponential time, there exists a function in NP that is Turing random self-reducible, but not truth-table random-self reducible.

In this paper we study the differences between many-one and Turing reductions in the context of completeness. Even though, it is standard to define completeness using many-one reductions, one can also define completeness using Turing reductions. A language L is *Turing complete* for a class $\mathcal{C}$ if L is in class $\mathcal{C}$ and every language in $\mathcal{C}$ Turing reduces to L. To capture the intuition that if a complete problem for a class $\mathcal{C}$ is “easy”, then the entire class is easy, Turing reductions are arguably more appropriate to define completeness. However, all known natural languages turn out to be complete under many-one reductions. This raises the following question: For a complexity class $\mathcal{C}$, is there a Turing complete language that is not many-one complete? This question was first posed by Ladner, Lynch, and Selman [18].

This question has been completely resolved for the complexity classes EXP and NEXP. Works of Ko and Moore [17] and Watanabe [27] showed that for EXP, almost all completeness notions are mutually different. Similar separation results are obtained for NEXP [5]. See survey articles [6, 14] for more details on these results.

For the case of NP, the progress has been very slow. The first result that achieves a separation between Turing and many-one completeness in NP, under a reasonable hypothesis, is due to Lutz and Mayordomo [20]. They showed that if NP does not have P-measure 0 (known as *measure hypothesis*), then Turing completeness for NP is different from many-one completeness. Ambos-Spies and Bentzien [2] achieved a finer separation under a weaker hypothesis known as *genericity hypothesis*. Subsequently, Turing and many-one completeness notions are shown to be different under even weaker hypotheses known as NP *machine hypothesis*, *bi-immunity hypothesis*, and *partial bi-immunity hypothesis* [13, 21, 22].

All of the above mentioned hypotheses are known as *almost everywhere hardness hypotheses*. Informally, these hypotheses assert that there exists a language in NP such that every algorithm that decides L must take more than subexponential time on *all but finitely many inputs*. Even though we believe that NP is subexponentially hard, we do not have any candidate languages in NP that are almost everywhere hard. All natural problems have an infinite set of instances that can be decided in polynomial time. Thus these hypotheses are considered “strong hypotheses”. It has been open whether a separation can be achieved using a *worst-case hardness* hypothesis (such as P $\neq$ NP, or NE $\neq$ E). The only partial result in this direction is due to Gu, Hitchcock, and Pavan [10] who showed that if there exist one-way permutations and there exists a language in NEEE $\cap$ co-NEEE that can not be solved in deterministic triple exponential time with logarithmic advice, then Turing completeness for NP differs from many-one completeness. Even though the latter hypothesis is a worst-case hardness hypothesis, the former is a average-case hardness hypothesis.

In this paper, we separate Turing completeness for NP from many-one completeness using a worst-case hardness hypothesis. This is the first result of this nature. Below is an informal statement of our result. Please see Section 3 for a more formal statement.

▶ **Main Theorem.** *Suppose there exist an* NEEXP *machine N accepting Σ^* and running in time $t(n)$ and a positive constant $\delta < 1$ such that no zero-error, probabilistic machine Z running in time $2^{t(n)^\delta}$ can compute accepting computation of N with non-trivial probability.*

Then there is a Turing complete language for NP *that is not truth-table complete for* NP*. Here we require that $t(n)$ is $2^{2^{n^c}}$ for some constant $c > 1$.*

The rest of the paper is organized as follows. Section 2 is the preliminaries section. In Section 3, we formally state our worst-case hardness hypothesis, and provide a proof of the separation theorem. Section 4 relates the hypothesis used in this paper to a few other hypotheses studied in the context of separating completeness notions.

2 Preliminaries

We use standard notions and definitions in complexity theory [3]. All languages are defined over the the binary alphabet $\Sigma = \{0, 1\}$, Σ^n denotes the set of all binary strings of length n. We use $|x|$ to denote the length of a string x. Non-deterministic double-exponential time is defined by $\mathrm{NEEXP} = \bigcup_{c>1} \mathrm{NTIME}(2^{2^{n^c}})$ and co-NEEXP is its complement class. We say that a non-deterministic machine is a NEEXP machine, if its runtime is bounded by $2^{2^{n^c}}$ for some $c > 1$. A language L is in ZPTIME$(t(n))$, if there is a probabilistic machine Z running in time $O(t(n))$ such that for every x, $\Pr[Z(x) = L(x)]$ is atleast $1/4$, and the probability that Z outputs an incorrect answer is zero. The machine Z may output $\perp$ with probability at most $3/4$.

▶ **Definition 1.** Suppose N is a non-deterministic machine accepting a language S. We say that a *$t(n)$-time bounded, zero-error, probabilistic machine computes accepting computations of N* if there exists a probabilistic machine Z such that

- For every $x \in S$, for every choice of random bits, the machine Z on input x either outputs a string from Σ^* or outputs the special symbol $\perp$.
- for every $x \in S$, $\Pr[Z(x)$ is an accepting computation of $N(x)] > 1/4$, and
- for every $x \in S$, $\Pr[Z(x) \neq \perp$ and is not an accepting computation of $N(x)] = 0$.

Our proof uses the notion of P-selective sets introduced by Selman [23].

▶ **Definition 2.** A set $S \subseteq \Sigma^*$ is P-*selective* if there is a polynomial time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for all strings $x, y \in \Sigma^*$, (1) $f(x, y) \in \{x, y\}$, and (2) if either of x and y is in S, then $f(x, y)$ is in S. The function f is called the P-*selector* of S.

The well-known example of P-selective sets are the *left-cut* sets $L(r) = \{x \mid x < r\}$, where r is an infinite binary sequence, and $<$ is the dictionary order with $0 < 1$. The following lemma is due to Toda [25].

▶ **Lemma 3.** *For every* P*-selective set L, there is a polynomial time algorithm that given any finite set of strings Q as input, outputs a sequence $x_1, \cdots, x_m$ such that $\{x_1, \cdots, x_m\} = Q$, such that for some integer p, $0 \le p \le m$, $Q \cap L = \{x_i \mid i \le p\}$ and $Q \cap \bar{L} = \{x_i \mid i > p\}$.*

Consider two languages A and B. A is *polynomial time Turing reducible* to B, denoted by $A \le_T^{\mathrm{P}} B$, if there is a polynomial time oracle Turing machine M such that $A = L(M^B)$. Note that M can make at most polynomially many queries to B and they can be *adaptive*.

The language A is *polynomial-time truth-table reducible* to B, denoted by $A \leq_{tt}^{\mathrm{P}} B$, if there is a pair of polynomial time computable functions $\langle f, g \rangle$ such that for every $x \in \Sigma^*$, (1) $f(x)$ is query set $Q = \{q_1, q_2, \cdots, q_k\}$ and (2) $x \in A \iff g(x, B(q_1), B(q_2), \cdots, B(q_k)) = 1$. We call f the *query generator* and g the *truth-table evaluator.* Given a polynomial time reducibility $\leq_r^{\mathrm{P}}$, a set B is $\leq_r^{\mathrm{P}}$*-complete* for NP if B is in NP and for every set $A \in$ NP, A is $\leq_r^{\mathrm{P}}$ reducible to B. Note that we only consider polynomial time reductions to define NP-completeness in this paper.

Notation. Let $\tau : \mathbb{N} \to \mathbb{N}$ be a function defined as $\tau(n) = 2^{2^n}$. The functions of the form $2^{2^{f(n)}}$, that are used in many places throughout this paper, are not visually appealing; from now we represent such functions as $\tau(f(n))$. Then $\tau(\delta f(n))$ represents $2^{2^{\delta f(n)}}$. We use $\tau^\epsilon(n)$ to denote $(\tau(n))^\epsilon$. Further, $\log^c n$ represents $(\log n)^c$.

3 Separation Theorem

In this section we prove the main result of this paper. First, we formally state our hypothesis.

Hypothesis W. *There exist a positive constant $\delta < 1$ and an* NEEXP *machine N_1 accepting Σ^* that runs in time $t(n)$ such that no $2^{t(n)^\delta}$-time bounded, zero-error, probabilistic machine can compute the accepting computations of N_1. Here $t(n) = 2^{2^{n^c}}$ for some constant $c > 1$.*

▶ **Theorem 4.** *If Hypothesis* W *holds, then there is a Turing complete language for* NP *that is not truth-table complete for* NP*.*

Before we provide a formal proof, we first describe proof outline. Our proof proceeds in four steps. Note that Hypothesis W is a "worst-case hardness hypothesis". This means that for every probabilistic, $2^{t(n)^\delta}$-time bounded, machine Z_1 there exists *infinitely many* inputs x such that the probability that $Z_1(x)$ computes an accepting computation of $N_1(x)$ is very small. This is equivalent to the following: there exist *infinitely many* input lengths n for which there exists *at least one string x of length n* so that the probability that $Z_1(x)$ is an accepting computation of $N_1(x)$ is very small. In the first step (Section 3.1), we amplify the hardness of N_1 and obtain an NEEXP machine N_2 with the following property: For every $2^{t(n)^\delta}$-time bounded, probabilistic machine Z_2, there exist *infinitely many input lengths n* at which *for every string x of length n* the probability that $Z_2(x)$ is an accepting computation of $N_2(x)$ is small.

In the second step (Section 3.2), we first define a padding function $pad : \Sigma^* \to \mathbb{N}$. Via standard padding arguments we obtain an NP-machine N running in time $p(n)$ that accepts a tally set $T = \{0^{pad(x)} \mid x \in \Sigma^*\}$. For $\ell \geq 0$, let $T_\ell = \{0^{pad(x)} \mid x \in \Sigma^\ell\}$. The NP-machine N has the following hardness property: For every $f(n)$-time bounded, probabilistic machine Z (for an appropriate choice of f) there exist infinitely many integers ℓ such that Z fails to compute accepting computations on *every string* from T_ℓ.

Using the NP-machine N, we define the Turing complete language L in step three (Section 3.3). The language L is formed by taking disjoint union of two NP languages L_1 and L_2. The language L_1 consists of tuple of the form $\langle x, a \rangle$ so that $x \in C$ (for some NP-complete language C), and a is an accepting computation of $N(0^n)$ (for some n that depends on x). In L_2, we encode accepting computations of N using a P-selective set. It follows that C can be Turing reduced to L by first obtaining an accepting computation of N (by making queries to L_2) and then by making one query to L_1. The idea of forming L_1 is borrowed from [21], and encoding accepting computations of an NP-machine as a P-selective sets is well known. For example see [11].

Finally, in step four (Section 3.4), we show that if L is truth-table complete, then there is a probabilistic machine Z such that for every ℓ there exists atleast one string in T_ℓ so that Z computes an accepting computation of N on that string with high probability. Using this, we in turn show that there exists a probabilistic machine Z_2 so that for every input length ℓ, there exists atleast one string $x \in \Sigma^\ell$ such that $Z_2(x)$ outputs an accepting computation of the NEEXP machine $N_2(x)$. This will be a contradiction. The most technical part of the proof lies in this step.

We now give proof details.

3.1 Hardness Amplification

The first step amplifies the hardness of the NEEXP machine N_1 from the hypothesis to obtain a new NEEXP machine N_2.

▶ **Lemma 5.** *Suppose that the hypothesis* W *holds. Then there exist an* NEEXP *machine N_2 accepting Σ^* and running in time $O(2^n\tau(n^c))$ and a constant $\beta < \delta$ such that for every probabilistic machine Z_2 that runs in time $\tau(\beta 2^{n^c})$, there exist* infinitely many input lengths *$n > 0$ such that for* every *$x \in \Sigma^n$,*

$$\Pr[Z_2(x) = \textit{ an accepting computation of } N_2(x)] \leq 1/4.$$

Proof. Let N_1 be the non-deterministic machine from Hypothesis W whose running time is bounded by $O(t(n))$, where $t(n) = \tau(n^c)$ (for some $c > 1$). Length of every accepting computation of $N_1(x)$ is bounded by $O(t(|x|))$. Consider a machine N_2 that behaves as follows: On an input x of length n, it runs $N_1(y)$ on every string y of length n (in a sequential manner). The running time of N_2 is $O(2^n \times t(n))$. Since N_1 accepts Σ^*, the machine N_2 also accepts Σ^*. We claim that N_2 has the required property.

Suppose not. Then there is a probabilistic machine Z_2 that runs in time $O(\tau(\beta 2^{n^c}))$ (for some $\beta < \delta$) such that for all but finitely many n, there exists a string $y_n \in \Sigma^n$ such that

$$\Pr[Z_2(y_n) = \text{ an accepting computation of } N_2(y_n)] > 1/4.$$

By the definition of N_2, the accepting computation of $N_2(x)$ encodes the accepting computation of $N_1(y)$ for every y whose length is same as the length of x. Consider a machine Z_1 that on any input x of length n behaves as follows:

> It runs $Z_2(y)$ on every y of length n. It verifies that the output of $Z_2(y)$ is an accepting computation of $N_2(y)$, and if the verification succeeds, then it extracts the accepting computation of $N_1(x)$ and outputs it. If $Z_2(y)$ does not output an accepting computation of $N_2(y)$, then Z_1 outputs $\perp$.

Let x be any input of length n. By our assumption, there exists a $y_n \in \Sigma^n$ such that $Z_2(y_n)$ outputs an accepting computation of $N_2(y_n)$ with probability at least 1/4. The above machine clearly runs $Z_2(y_n)$ on input x. Since an accepting computation of $N_1(x)$ can be retrieved from an accepting computation of $N_2(y_n)$, the above machine outputs an accepting computation of $N_1(x)$. Thus for all but finitely many n, for every $x \in \Sigma^n$, Z_1 outputs an accepting computation of $N_1(x)$ with probability at least 1/4. The running time of Z_1 is clearly $O(2^n \times \tau(\beta 2^{n^c}))$, which is less than $\tau(\delta 2^{n^c})$ (as $\beta < \delta$). This contradicts Hypothesis W. ◀

3.2 Defining an NP machine

In this section, we define an NP machine N from the above NEEXP machine N_2. Fix $\epsilon < \beta$. Consider the following padding function $pad : \Sigma^* \to \mathbb{N}$, defined by

$$pad(x) = \lfloor \tau^{\epsilon}(\log^c r_x) \rfloor,$$

where r_x is the rank of string x in the standard lexicographic order of Σ^*, so that $2^{\ell} - 1 \leq r_x \leq 2^{\ell+1} - 2$, for every $x \in \Sigma^{\ell}$. Note that pad is 1-1 and so $pad^{-1}(n)$ (if exists) is well defined. To keep the calculation simple, we drop the floors henceforth. Now we define the following tally language based on the padding function:

$$T = \left\{ 0^{pad(x)} \mid x \in \Sigma^* \right\}.$$

Our NP machine N that accepts a tally language behaves as follows:

> On input 0^m, it computes $x = pad^{-1}(m)$. Upon finding such x, it runs $N_2(x)$. If no such x is found, then N rejects.

Note that $|x| < (\log\log m^{2/\epsilon})^{1/c}$. So running time of N is bounded by $m^{3/\epsilon}$. Thus N is an NP machine. Note that N accepts the tally language T.

3.3 Turing-complete language

At this point, we are ready to define the language L in NP that we prove to be Turing complete, but not truth-table complete for NP.

Let L_T be the range of the padding function pad.

$$L_T = \{\tau^{\epsilon}(\log^c i) \mid i \in \mathbb{N}\}.$$

By definition, N accepts only those tally strings whose length is in the set L_T. We use n_i to denote $pad(i)$. Given a length $n \in L_T$, define a_n to be the lexicographically maximum accepting computation of $N(0^n)$. Let a be the infinite binary string $a_{n_1}a_{n_2}a_{n_3}\cdots$ where $n_i \in L_T$ and $n_1 < n_2 < n_3 < \cdots$. Let $|a_n|$ denotes the length of the accepting computation a_n. Let SAT$'$ consist of the SAT formulas with lengths only in L_T, *i.e.*,

$$\mathrm{SAT}' = \mathrm{SAT} \cap \{x \in \Sigma^* \mid |x| \in L_T\}.$$

Since there exists a polynomial p such that $n_{i+1} \leq p(n_i)$, it can be shown via padding that SAT many-one reduces to SAT$'$ and thus SAT$'$ is NP-complete.

We define L_1 and L_2 as follows:

$$L_1 = \left\{ \langle \phi, u \rangle \mid |\phi| = n,\ u \text{ is an accepting computation of } N \text{ on } 0^n,\ \phi \in \mathrm{SAT}' \right\}$$

and

$$L_2 = L(a) = \{z \mid z < a\},$$

where $<$ is the dictionary order with $0 < 1$. Then our Turing-complete language L is the disjoint union of L_1 and L_2, *i.e.*,

$$L = L_1 \uplus L_2 = 0L_1 \cup 1L_2.$$

Note that both L_1 and L_2 are in NP, and so is L.

▶ **Lemma 6.** *L is $\leq_T^{\mathrm{P}}$-complete for* NP.

Proof. Reduce SAT$'$ to L: On input ϕ of length n, make adaptive queries to L_2 to find a_n. Accept ϕ if and only if $\langle \phi, a_n \rangle \in L_1$. ◀

3.4 L is not truth-table complete

In this section, we show that L is not truth-table complete for NP. Before we proceed with the proof, we provide the intuition behind the proof. Suppose that L is truth-table complete. We achieve a contradiction by exhibiting a procedure to compute accepting computations of NEEXP machine N_2. Since the NP-machine N is padded version of N_2, it suffices to compute the accepting computations of N. We partition T into sets $T_1, T_2, \cdots$, where $T_\ell = \{0^{pad(x)} \mid x \in \Sigma^\ell\}$. Clearly, $|T_\ell| = 2^\ell$ and $T = \bigcup_\ell T_\ell$. Note that an accepting computation of $N_2(x)$ can be computed by computing an accepting computation of $N(0^{pad(x)})$, and if $|x| = \ell$, then $0^{pad(x)} \in T_\ell$.

Recall that N_2 has the following property: For every probabilistic machine Z_2 that attempts to compute its accepting computations, there exist infinitely many input lengths ℓ and Z_2 fails on every string at those lengths. Informally, this translates to the following hardness property of N: For every probabilistic machine Z that attempts to compute accepting computations of N, there exist infinitely many integers ℓ such that Z fails on every string from T_ℓ. Thus to achieve a contradiction, it suffices to exhibit a probabilistic procedure Z such that for all but finitely many ℓ, Z outputs an accepting computation of $N(0^n)$ for some $0^n \in T_\ell$, with non-negligible probability. We will now (informally) describe how to compute accepting computations of N.

For the sake of simplicity, let us first assume that the NP machine N has exactly one accepting computation on every input from T. The first task is to define a set S that encodes the accepting computations of the machine N. One way to define S as

$$S = \{\langle 0^n, i\rangle \mid i\text{th bit of accepting computation of } N(0^n) \text{ is } 1\}.$$

Since we assumed that N has exactly one accepting computation, deciding S is equivalent to computing accepting computations of N. Since S is in NP, there is a truth-table reduction from S to L. We make another simplifying assumption that all queries are made to L_1 part of L. Consider an input $\langle 0^n, i\rangle$ where $0^n \in T_\ell$ (for some $\ell > 0$). All the queries produced on this input are of the form $\langle \phi, u\rangle$. It is easy to check if u is an accepting computation of $N(0^m)$ for some m. If u is not an accepting computation, then $\langle \phi, u\rangle$ does not belong to L, and thus it is easy to decide the membership of $\langle 0^n, i\rangle$ in S. Suppose that u is an accepting computation of $N(0^m)$ for some m. Then there are two cases. First case is the "short query" case, where m is much smaller than n. In this case $\langle \phi, u\rangle$ is in L_1 only when $|\phi|$ equals m and $\phi \in \text{SAT}'$. Since $m << n$, we can decide whether $\phi \in \text{SAT}'$ using a brute force algorithm in time $O(2^m)$, this in turn enables us to decide the membership of $\langle 0^n, i\rangle$ in S. Thus if all the queries are small, we can decide the memberships of $\langle 0^n, i\rangle$ (for all i), and thus can compute accepting computation of $N(0^n)$. The second case is the "large query" case: Suppose that for some query, m is not much smaller than n. In this case, we are in the following scenario: The reduction outputs accepting computation of $N(0^m)$ and m is somewhat large. In this case, we argue that for an appropriate choice of n, 0^m also lies in T_ℓ. This will enable us to design a procedure that outputs accepting computation of some string from T_ℓ. This is the gist of the proof.

The above argument assumed that N has exactly one accepting computation, which is not true in general. We get around this problem by applying Valiant-Vazirani lemma [26] to isolate one accepting computation. Thus our language S will involve the use of isolation lemma. It is also very much possible that the reduction makes queries to L_2 also. Recall that L_2 is a P-selective set and it is known that if an NP-language A reduces to a P-selective set, then A must be "easy" [25, 23]. We use this in combination with the above mentioned approach. A technically involved part is to define the correct notion of "small" and "large" queries. There is a fine interplay among the choice of pad function, notion of small query, and the runtime of probabilistic machine that computes the accepting computations of N. We now provide a formal proof.

▶ **Lemma 7.** *L is not $\leq_{tt}^{\mathrm{P}}$-complete for* NP.

Proof. For the sake of contradiction, assume that L is truth-table complete for NP. Consider the following set S.

$$S = \{\langle 0^n, k, r_1, r_2, \ldots, r_k, i\rangle \mid n \in L_T, 1 \leq k \leq |a_n|, r_i \in \Sigma^{|a_n|}, \text{ there is a } u \text{ such that } u \text{ is an accepting computation of } N(0^n), i\text{th bit of } u = 1, \text{ and } u \cdot r_1 = u \cdot r_2 = \cdots = u \cdot r_k = 0\},$$

where $u \cdot r_i$ denotes the inner product of u and r_i, for all i, over GF[2].

It is easy to see that S is in NP. Since L is $\leq_{tt}^{\mathrm{P}}$-complete for NP, S is $\leq_{tt}^{\mathrm{P}}$ reducible to L via polynomial time computable functions $\langle g, h\rangle$, where g is the query generator and h is the truth-table evaluator. Since g is polynomial-time computable, there exists a constant $b > 0$ such that every query generated by it is of length at most n^b.

At this point, our goal is to compute an accepting computation of N. We start with the following algorithm $\mathcal{A}$ that classifies all the queries of the query generator into two sets, "Large Query" and "Small Query".

1. Input 0^n, where $n = \tau^{\epsilon}(\log^c i)$ for some $i \in \mathbb{N}$. Clearly, $n \in L_T$.
2. For $1 \leq j \leq n^2$ repeat the following:
   - Pick k^j uniformly at random from $\{1, \cdots, |a_n|\}$.
   - Pick each of $r_1^j, r_2^j, \ldots, r_{k_j}^j$ uniformly at random from $\Sigma^{|a_n|}$.
3. Let Q^j be the set of queries generated by g on inputs $\langle 0^n, k^j, r_1^j, \cdots, r_{k_j}^j, i\rangle$, $1 \leq i \leq |a_n|$. Compute Q^j for $1 \leq j \leq n^2$ and set $Q = \bigcup_j Q^j$. Note that the length of each query is bounded by n^b.
4. Partition Q into two sets Q_1 and Q_2 such that Q_1 is the set of all queries to L_1 and Q_2 is the set of all queries to L_2.
5. If Q_1 contains a query $\langle \phi, u_t\rangle$ for some t, where u_t is an accepting computation of $N(0^t)$ and
$$t > \tau^{\epsilon}(((\log\log n^{b/\epsilon})^{1/c} - 1)^c),$$
then print u_t, output "Large Query", and halt.
6. Otherwise, output "Small Query" and halt.

It is clear that the algorithm $\mathcal{A}$ runs in time polynomial in n.

Before we give our probabilistic algorithm to compute the accepting computations of N, we bound the probabilities of certain events of interest. T is partitioned into sets $T_1, T_2, \cdots$ each of cardinality 2^{ℓ}, where

$$T_{\ell} = \left\{0^{\tau^{\epsilon}(\log^c r_x)} \mid x \in \Sigma^{\ell}\right\}.$$

Fix $\ell > 0$. For a fixed $0^n \in T_{\ell}$ and j, $1 \leq j \leq n^2$, let $\mathcal{E}_{n,j}$ denote the following event:

There exists *exactly one* u such that
- u is an accepting computation on $N(0^n)$,
- $u \cdot r_1^j = u \cdot r_2^j = \cdots = u \cdot r_{k_j}^j = 0$.

By Valiant-Vazirani, we have that $\Pr[\mathcal{E}_{n,j}] \geq \frac{1}{n^2}$. Let $\mathcal{E}_n$ denote the event that for some j, $1 \leq j \leq n^2$, $\mathcal{E}_{n,j}$ occurs. The probability of $\mathcal{E}_n$ is at least $1 - \frac{1}{2^{n^2}}$. Finally, let $\mathcal{E}_\ell$ denote the event that for every $0^n \in T_\ell$, the event $\mathcal{E}_n$ occurs. Again, we have that $\Pr[\mathcal{E}_\ell] \geq 1 - \frac{1}{2^\ell}$.

Thus for every ℓ, the probability that the event $\mathcal{E}_\ell$ occurs is very high. Fix an ℓ. From now on, we assume that the event $\mathcal{E}_\ell$ has occurred.

Now our goal is to arrive at the machine that computes an accepting computation of atleast one string from T_ℓ. For this we will analyze the behavior of the above algorithm on a specific string $0^{V_\ell} \in T_\ell$, where

$$V_\ell = \tau^{\epsilon/b}(\log^c(2^{\ell+1} - 2)).$$

We stress that this *unique* string 0^{V_ℓ} depends only on the length ℓ. When we run algorithm $\mathcal{A}$ on 0^{V_ℓ}, either it outputs “Large Query” on it, or it outputs “Small Query”.

▶ **Lemma 8** (Key Lemma). *One of the following holds.*

1. *If $\mathcal{A}$ outputs “Small Query” on 0^{V_ℓ}, then there is an algorithm $\mathcal{B}_1$ that on input 0^{V_ℓ} runs in time polynomial in $\tau(\epsilon 2^{((\log\log V_\ell{}^{b/\epsilon})^{1/c}-1)^c})$, and correctly outputs an accepting computation of $N(0^{V_\ell})$.*
2. *If $\mathcal{A}$ outputs “Large Query” on 0^{V_ℓ}, there exist an algorithm $\mathcal{B}_2$ such that for every string in T_ℓ it runs in time polynomial in V_ℓ, and there exists a $0^t \in T_\ell$ for which $\mathcal{B}_2(0^t)$ outputs an accepting computation of $N(0^t)$.*

Due to the lack of space, we defer the proof of this lemma to the full version of the paper. Now we complete the proof of main theorem by describing a probabilistic machine that computes accepting computation of the NEEXP machine N_2.

Computing accepting computations of N_2

Remember that we defined our NEEXP machine N_2 in Lemma 5. Now consider the probabilistic machine Z_2 that does the following on input $x \in \Sigma^\ell$:

1. Compute V_ℓ. Run $\mathcal{A}$ on 0^{V_ℓ}.
2. If $\mathcal{A}(0^{V_\ell})$ outputs “Small Query”,
   - Verify if $x = pad^{-1}(V_\ell)$. If it is, then run $\mathcal{B}_1$ on 0^{V_ℓ} and if it outputs an accepting computation of $N(0^{V_\ell})$, then output that accepting computation. This is also the accepting computation of $N_2(x)$.
3. If $\mathcal{A}(0^{V_\ell})$ outputs “Large Query”, do the following:
   - For every string 0^i in T_ℓ, run the algorithm $\mathcal{B}_2$ on it. If it outputs the accepting computation of $N(0^t)$ for some 0^t, then verify if $x = pad^{-1}(0^t)$. If it is, then output that accepting computation. This is also the accepting computation of $N_2(x)$.

We analyze the behavior of Z_2 under the assumption that the event $\mathcal{E}_\ell$ happens. Recall that this happens with very high probability. If $\mathcal{A}(0^{V_\ell})$ outputs “Small Query”, then by part (1) of Lemma 8, $\mathcal{B}_1$ outputs an accepting computation of $N(0^{V_\ell})$. Note that every accepting computation of $N(0^{V_\ell})$ is an accepting computation of $N_2(pad^{-1}(V_\ell))$. Since $pad^{-1}(V_\ell)$ is of length ℓ, there exists a string $x \in \Sigma^\ell$, on which Z_2 outputs an accepting computation of $N_2(x)$. Now consider the case where $\mathcal{A}(0^{V_\ell})$ outputs “Large Query”, then by part (2) of Lemma 8, there exists a $0^t \in T_\ell$ such that $\mathcal{B}_2(0^t)$ outputs an accepting computation of $N(0^t)$. Thus Z_2 will find that 0^t through iteration. Similarly, $pad^{-1}(0^t) \in T_\ell$ is of length ℓ, thus there exists a x in Σ^ℓ on which Z_2 outputs an accepting computation of $N_2(x)$. Thus Z_2 always outputs an accepting computation of atleast one string x from Σ^ℓ.

We will now bound the runtime of Z_2. This is bounded by runtime of $A(0^{V_\ell})$, plus the runtime of $\mathcal{B}_1(0^{V_\ell})$, and the time taken in step 3 of the above algorithm. By part (1) of Lemma 8, the runtime of $\mathcal{B}_1(0^{V_\ell})$ is $\tau^d(\epsilon 2^{((\log\log V_\ell{}^{b/\epsilon})^{1/c}-1)^c})$ for some constant $d > 0$, which is bounded by

$$\tau^d(\epsilon 2^{((\log\log V_\ell{}^{b/\epsilon})^{1/c}-1)^c}) = \tau^d(\epsilon 2^{((\log^c(2^{\ell+1}-2))^{1/c}-1)^c}) = \tau^d(\epsilon 2^{((\log(2^\ell-1))^c}) < \tau^d(\epsilon 2^{\ell^c}).$$

Let p be a constant such that $\mathcal{A}(0^{V_\ell})$ runs in time V_ℓ^p and $\mathcal{B}_2(0^i)$, $0^i \in T_\ell$, runs in time V_ℓ^p. Step 3 runs $\mathcal{B}_2$ on every string from T_ℓ, and there are 2^ℓ strings in T_ℓ. Thus the combined runtime of $\mathcal{A}(0^{V_\ell})$ in step 1 and step 3 is bounded by

$$2^{\ell+1}V_\ell^p = 2^{\ell+1}\tau^{p\epsilon/b}(\log^c(2^{\ell+1}-2)) \leq 2^{\ell+1}\tau^{p\epsilon/b}((\ell+1)^c) \leq \tau^q((\ell+2)^c)$$

for some constant $q > p$. Thus the total running time of Z_2 is bounded by $\tau(\beta 2^{\ell^c})$, as $\beta > \epsilon$.

Thus for all but finitely many ℓ, the machine Z_2 computes an accepting computation of $N_2(x)$ for atleast one string x from Σ^ℓ with non-trivial probability. This contradicts the hardness of NEEXP machine N_2 in Lemma 5. This completes the proof of Lemma 7. ◀

This also completes the proof the main theorem.

4 Power of the hypothesis

In this section, we show some results that explain the power of Hypothesis W and also compare it to some of the previously studied hypotheses that are used to separate NP-completeness notions. All proofs in this section appear in the full version of the paper.

Even though Hypothesis W talks about the difficulty of computing accepting computations of NEEXP machines, our first result states that it can be related to the hardness of the complexity class NEEXP ∩ co-NEEXP.

Hypothesis 2. There exist constants $c > 1$ and $0 < \delta < 1$ such that NTIME$(t(n))$ ∩ co-NTIME$(t(n)) \nsubseteq$ ZPTIME$(2^{t(n)^\delta})$, for $t(n) = 2^{2^{n^c}}$.

Now we show that our hypothesis follows from this worst-case separation hypothesis.

▶ **Proposition 1.** *Hypothesis 2 implies Hypothesis W.*

Pavan and Selman [21] showed that the NP-completeness notions differ under the following hypothesis.

Hypothesis 3 (NP-machine Hypothesis)**.** There exist an NP machine N accepting 0^* and $\beta > 0$ for every 2^{n^β}-time bounded deterministic algorithm M, $M(0^n)$ does not output an accepting computation of $N(0^n)$ *for all but finitely many n.*

Note that the hypothesis requires that every machine that attempts to compute accepting computations of N must fail on *all but finitely many inputs.* This type of hardness hypothesis is called "almost everywhere hardness hypothesis". In contrast, Hypothesis W requires that every machine that attempts to compute accepting computations of the NEEXP machine must fail on only *infinitely many strings.*

Ideally, we would like to show that NP-machine hypothesis implies Hypothesis W. However, NP-machine hypothesis concerns with hardness against deterministic algorithms, whereas Hypothesis W concerns with hardness against probabilistic algorithms. If we assume well-accepted derandomization hypotheses, we can show Hypothesis W is weaker than the NP-machine hypothesis.

▶ **Proposition 2.** *Suppose that* ZPP = P*. If* NP*-machine hypothesis holds, then Hypothesis W holds.*

Lutz and Mayordomo [20] achieved the separation of NP-completeness notions under the *Measure Hypothesis*. Hitchcock and Pavan [12] showed that *Measure hypothesis* implies the above NP-machine hypothesis. Thus we have the following.

▶ **Proposition 3.** *Suppose that* ZPP = P*. Measure hypothesis implies Hypothesis W.*

Pavan and Selman [22] showed that if NP-contains 2^{n^ϵ}-*bi-immune* sets, then completeness in NP differ. Informally, the hypothesis means the following: There is a language L in NP such that every 2^{n^ϵ}-time bounded algorithm that attempts to decide L must fail on *all but finitely many strings*. Thus this hypothesis concerns with almost-everywhere hardness, whereas Hypothesis W concerns with worst-case hardness. We are not able to show that the bi-immunity hypothesis implies Hypothesis W (even under the assumption ZPP = P). However, we note that if NP ∩ co-NP has bi-immune sets, then Hypothesis W follows. Pavan and Selman [21] showed that if NP ∩ co-NP has a DTIME(2^{n^ϵ})-*bi-immune* set, then NP-machine hypothesis follows.

▶ **Proposition 4.** *Suppose that* ZPP = P*. If* NP ∩ co-NP *has a* DTIME(2^{n^ϵ})*-bi-immune set, then Hypothesis W holds.*

5 Conclusions

This paper, for the first time, shows that Turing completeness for NP can be separated from many-one completeness under a worst-case hardness hypothesis. Our hypothesis concerns with hardness of nondeterministic, double exponential time. An obvious question is to further weaken the hypothesis. Can we achieve the separation under the assumption that there exists a language in NE that can not be solved in deterministic/probabilistic time $O(2^{\delta 2^n})$?

References

1 S. Aida, R. Schuler, T. Tsukiji, and O. Watanabe. On the difference between polynomial-time many-one and truth-table reducibilities on distributional problems. In *18th International Symposium on Theoretical Aspects of Computer Science*, 2001.

2 K. Ambos-Spies and L. Bentzien. Separating NP-completeness under strong hypotheses. *Journal of Computer and System Sciences*, 61(3):335–361, 2000.

3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

4 L. Babai and S. Laplante. Stronger separations ofor random-self-reducibility, rounds, and advice. In *14th IEEE Conference on Computational Complexity*, pages 98–104, 1999.

5 H. Buhrman, S. Homer, and L. Torenvliet. Completeness notions for nondeterministic complexity classes. *Mathematical Systems Theory*, 24:179–200, 1991.

6 H. Buhrman and L. Torenvliet. On the structure of complete sets. In *9th IEEE Annual Conference on Structure in Complexity Theory*, pages 118–133, 1994.

7 S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

8 J. Feigenbaum, L. Fortnow, C. Lund, and D. Spielman. The power of adaptiveness and additional queries in random-self-reductions. In *Proc. 7th Annual Conference on Structure in Complexity Theory*, pages 338–346, 1992.

9 J. Feigenbaun, L. Fortnow, S. Laplante, and A. Naik. On coherence, random-self-reducibility, and self-correction. In *Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity*, pages 224–232, 1996.

10 X. Gu, J. Hitchcock, and A. Pavan. Collapsing and separating completeness notions under average-case and worst-case hypotheses. *Theory of Computing Systems*, 51(2):248–265, 2011.

11 E. Hemaspaandra, A. Naik, M. Ogiwara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.

12 J. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. *Computational Complexity*, 17(1):119–146, 2008.

13 J. Hitchcock, A. Pavan, and N. V. Vinodchandran. Partial bi-immunity, scaled dimension and np-completeness. *Theory of Computing Systems*, 42(2):131–142, 2008.

14 S. Homer. Structural properties of complete problems for exponential time. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 135–153. Springer-Verlag, 1997.

15 D. W. Juedes and J. H. Lutz. The complexity and distribution of hard problems. *SIAM Joutnal on Computing*, 24:279–295, 1995.

16 R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

17 K. Ko and D. Moore. Completeness, approximation and density. *SIAM Journal on Computing*, 10(4):787–796, Nov. 1981.

18 R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

19 L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. English translation of original in *Problemy Peredaci Informacii.*

20 J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theoretical Computer Science*, 164:141–163, 1996.

21 A. Pavan and A. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.

22 A. Pavan and A. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188:116–126, 2004.

23 A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.

24 S. Tang, B. Fu, and T. Liu. Exponential time and subexponential time sets. *Theoretical Computer Science*, 115(2):371–381, 1993.

25 S. Toda. On polynomial-time truth-table reducibilities of intractable sets to P-selective sets. *Mathematical Systems Theory*, 24(2):69–82, 1991.

26 L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

27 O. Watanabe. A comparison of polynomial time completeness notions. *Theoretical Computer Science*, 54:249–265, 1987.

Constructing Small Tree Grammars and Small Circuits for Formulas

Danny Hucke, Markus Lohrey, and Eric Noeth

University of Siegen, Germany
{hucke,lohrey,eric.noeth}@eti.uni-siegen.de

Abstract

It is shown that every tree of size n over a fixed set of σ different ranked symbols can be decomposed into $O(\frac{n}{\log_\sigma n}) = O(\frac{n \log \sigma}{\log n})$ many hierarchically defined pieces. Formally, such a hierarchical decomposition has the form of a straight-line linear context-free tree grammar of size $O(\frac{n}{\log_\sigma n})$, which can be used as a compressed representation of the input tree. This generalizes an analogous result for strings. Previous grammar-based tree compressors were not analyzed for the worst-case size of the computed grammar, except for the top dag of Bille et al., for which only the weaker upper bound of $O(\frac{n}{\log^{0.19} n})$ for unranked and unlabelled trees has been derived. The main result is used to show that every arithmetical formula of size n, in which only $m \le n$ different variables occur, can be transformed (in time $O(n \log n)$) into an arithmetical circuit of size $O(\frac{n \cdot \log m}{\log n})$ and depth $O(\log n)$. This refines a classical result of Brent, according to which an arithmetical formula of size n can be transformed into a logarithmic depth circuit of size $O(n)$. Missing proofs can be found in the long version [14].

1998 ACM Subject Classification E.4 Data compaction and compression

Keywords and phrases grammar-based compression, tree compression, arithmetical circuits

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.457

1 Introduction

Grammar-based compression has emerged to an active field in string compression during the past 20 years. The idea is to represent a given string s by a small context-free grammar that generates only s; such a grammar is also called a *straight-line program*, briefly SLP. For instance, the word $(ab)^{1024}$ can be represented by the SLP with the productions $A_0 \to ab$ and $A_i \to A_{i-1}A_{i-1}$ for $1 \le i \le 10$ (A_{10} is the start symbol). The size of this grammar is much smaller than the size (length) of the string $(ab)^{1024}$. In general, an SLP of size n (the size of an SLP is usually defined as the total length of all right-hand sides of the productions) can produce a string of length $2^{\Omega(n)}$. Hence, an SLP can be seen indeed as a succinct representation of the generated string. The goal of grammar-based string compression is to construct from a given input string s a small SLP that produces s. Several algorithms for this have been proposed and analyzed. Prominent grammar-based string compressors are for instance LZ78, RePair, and BISECTION, see [7] for more details.

To evaluate the compression performance of a grammar-based compressor $\mathcal{C}$, two different approaches can be found in the literature: A first approach is to analyze the size of the SLP produced by $\mathcal{C}$ for an input string x compared to the size of a smallest SLP for x. This leads to the approximation ratio for $\mathcal{C}$, see [7] for a formal definition. It is known that unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial time grammar-based compressor that produces for every string x an SLP of size strictly smaller than $8569/8568 \cdot g(x)$, where $g(x)$ is the size of a smallest SLP for x [7]. The best known polynomial time grammar-based compressors have an approximation ratio of $\mathcal{O}(\log(n/g))$, where g is the size of a smallest SLP for the input string, and each of them works in linear time; see [22] for references.

Another approach is to analyze the maximal size of SLPs produced by $\mathcal{C}$ on strings of length n over the alphabet Σ (the size of Σ is considered to be a constant larger than one in the further discussion). An information-theoretic argument shows that for almost all strings of length n (up to an exponentially small part) the smallest SLP has size $\Omega(\frac{n}{\log n})$. Explicit examples of strings for which the smallest SLP has size $\Omega(\frac{n}{\log n})$ result from de Bruijn sequences; see Section 2. On the other hand, many grammar-based compressors produce for every string of length n an SLP of size $O(\frac{n}{\log n})$. This holds for instance for the above mentioned LZ78, RePair, and BISECTION, and in fact for all compressors that produce so-called irreducible SLPs [16]. This fact is used in [16] to construct universal string compressors based on grammar-based compressors.

In this paper, we follow the latter approach, but for trees instead of strings. A tree in this paper is always a rooted ordered tree over a ranked alphabet, i. e., every node is labelled with a symbol and the rank of this symbol is equal to the number of children of the node. In [6], grammar-based compression was extended from strings to trees. For this, linear context-free tree grammars were used. Linear context-free tree grammars that produce only a single tree are also known as tree straight-line programs (TSLPs) or straight-line context-free tree grammars (SLCF tree grammars). TSLPs generalize dags (directed acyclic graphs), which are widely used as a compact tree representation. Whereas dags only allow to share repeated subtrees, TSLPs can also share repeated internal tree patterns.

Several grammar-based tree compressors were developed in [1, 6, 15, 23]. The algorithm from [15] achieves an approximation ratio of $O(\log n)$ (for a constant set of node labels). On the other hand, for none of the above mentioned compressors it is known, whether for any input tree with n nodes the size of the output grammar is bounded by $O(\frac{n}{\log n})$, as it is the case for many grammar-based string compressors. Recently, it was shown that the so-called *top dag* of an unranked and unlabelled tree of size n has size $O(\frac{n}{\log^{0.19} n})$ [3]. The top dag can be seen as a slight variant of a TSLP for an unranked tree.

In this paper, we present a grammar-based tree compressor that transforms a given node-labelled ranked tree of size n with σ different node labels into a TSLP of size $O(\frac{n}{\log_\sigma n})$ and depth $O(\log n)$, where the depth of a TSLP is the depth of the corresponding derivation tree. In particular, for an unlabelled binary tree we get a TSLP of size $O(\frac{n}{\log n})$. Our compressor is an extension of the BISECTION algorithm [17] from strings to trees and works in two steps (the following outline works only for binary trees, but it can be easily adapted to trees of higher ranks): In the first step, we hierarchically decompose the tree into pieces of roughly equal size, using a well-known lemma from [19]. But care has to be taken to bound the ranks of the nonterminals of the resulting TSLP. As soon as we get a tree with three holes during the decomposition (which corresponds in the TSLP to a nonterminal of rank three) we do an intermediate step that decomposes the tree into two pieces having only two holes each. This may involve an unbalanced decomposition. On the other hand, such an unbalanced decomposition is only necessary in every second step. This trick to bound the number of holes by three was used by Ruzzo [25] in his analysis of space-bounded alternation.

The TSLP produced in the first step can be identified with its derivation tree. Thanks to the fact that all nonterminals have rank at most three, we can encode the derivation tree by a tree with $O(\sigma)$ many labels. Moreover, this derivation tree is weakly balanced in the following sense. For each edge (u, v) in the derivation tree such that both u and v are internal nodes, the derivation tree is balanced at u or v. These facts allow us to show that the minimal dag of the derivation tree has size at most $O(\frac{n}{\log_\sigma n})$. The nodes of this dag are the nonterminals of our final TSLP. The running time of our algorithm is in $O(n \log n)$.

Our size bound $O(\frac{n}{\log_\sigma n})$ does not contradict the information-theoretic lower bound: Consider for instance unlabelled ordered trees. When encoding a TSLP of size m into a bit string, we get an additional $\log(m)$-factor. Hence, a TSLP of size $O(\frac{n}{\log n})$ is encoded by a bit string of size $O(n)$, which is the information-theoretic bound (the exact bound is $2n - o(n)$).

It is important to note that our size bound $O(\frac{n}{\log_\sigma n})$ only holds for ranked trees and does not directly apply to unranked trees (that are, for instance, the standard tree model for XML). To overcome this limitation, one can transform an unranked tree of size n into its first-child-next-sibling encoding [18, Paragraph 2.3.2], which is a ranked tree of size n. Then, the first-child-next-sibling encoding can be transformed into a TSLP of size $O(\frac{n}{\log_\sigma n})$.

Our main result has an interesting application for the classical problem of transforming formulas into small circuits. Spira [26] has shown that for every Boolean formula of size n there exists an equivalent Boolean circuit of depth $O(\log n)$ and size $O(n)$. Brent [4] extended Spira's theorem to formulas over arbitrary semirings and moreover improved the constant in the $O(\log n)$ bound. Subsequent improvements that mainly concern constant factors can be found in [5]. An easy corollary of our $O(\frac{n}{\log_\sigma n})$ bound for TSLPs is that for every (not necessarily commutative) semiring (or field), every formula of size n, in which only $m \leq n$ different variables occur, can be transformed into a circuit of depth $O(\log n)$ and size $O(\frac{n \cdot \log m}{\log n})$. Hence, we refine the size bound from $O(n)$ to $O(\frac{n \cdot \log m}{\log n})$ (Theorem 9). Another interesting point of our formula-to-circuit conversion is that most of the construction (namely the construction of a TSLP for the input formula) is purely syntactic. The remaining part (the transformation of the TSLP into a circuit) is straightforward.

Related work. Several papers deal with algorithmic problems on trees that are succinctly represented by TSLPs, see [22] for a survey. Among other problems, equality checking and the evaluation of tree automata can be done in polynomial time for TSLPs.

It is interesting to compare our $O(\frac{n}{\log_\sigma n})$ bound with the known bounds for dag compression. A counting argument shows that for almost all unlabelled binary trees, the size of a smallest TSLP is $\Omega(\frac{n}{\log n})$, and hence (by our main result) $\Theta(\frac{n}{\log n})$. This implies that the average size of the minimal TSLP, where the average is taken for the uniform distribution on unlabelled binary trees of size n, is $\Theta(\frac{n}{\log n})$ as well. In contrast, the size of the minimal dag for trees of size n is $\Theta(n/\sqrt{\log n})$ on average [11] but n in the worst case.

2 Strings and Straight-Line Programs

Before we come to grammar-based tree compression, let us briefly discuss grammar-based string compression. A *straight-line program*, briefly SLP, is a context-free grammar that produces a single string. Formally, it is a tuple $\mathcal{G} = (N, \Sigma, P, S)$, where N is a finite set of nonterminals, Σ is a finite set of terminal symbols ($\Sigma \cap N = \emptyset$), $S \in N$ is the start nonterminal, and P is a finite set of productions of the form $A \to w$ for $A \in N$, $w \in (N \cup \Sigma)^*$ such that: (i) if $(A \to u), (A \to v) \in P$ then $u = v$, and (ii) the binary relation $\{(A, B) \in N \times N \mid (A \to w) \in P,\ B \text{ occurs in } w\}$ is acyclic. Every nonterminal $A \in N$ produces a unique string $\mathrm{val}_{\mathcal{G}}(A) \in \Sigma^*$. The string defined by $\mathcal{G}$ is $\mathrm{val}(\mathcal{G}) = \mathrm{val}_{\mathcal{G}}(S)$. The size of the SLP $\mathcal{G}$ is $|\mathcal{G}| = \sum_{(A \to w) \in P} |w|$, where $|w|$ is the length of w.

Let σ be the size of the terminal alphabet Σ. It is well-known that for every string $x \in \Sigma^*$ of length n there exists an SLP $\mathcal{G}$ of size $O(n/\log_\sigma n)$ such that $\mathrm{val}(\mathcal{G}) = x$, see e.g. [16]. On the other hand, an information-theoretic argument shows that for almost all strings of length n, the smallest SLP has size $\Omega(n/\log_\sigma n)$. For SLPs, one can, in contrast to other models like Boolean circuits, construct explicit strings that achieve this worst-case bound:

▶ **Proposition 1.** *Let Σ be an alphabet of size σ. For every $n \geq \sigma^2$, one can construct in time* $\mathrm{poly}(n, \sigma)$ *a string $s_{\sigma,n} \in \Sigma^*$ of length n such that every SLP for $s_{\sigma,n}$ has size $\Omega(n/\log_\sigma n)$.*

Proof. Let $r = \lceil \log_\sigma n \rceil \geq 2$. The sequence $s_{\sigma,n}$ is in fact a prefix of a de Bruijn sequence [9]. Let $x_1, \dots x_{\sigma^{r-1}}$ be a list of all words from Σ^{r-1}. Construct a directed graph by taking these strings as vertices and drawing an a-labelled edge ($a \in \Sigma$) from x_i to x_j if $x_i = bw$ and $x_j = wa$ for some $w \in \Sigma^{r-2}$ and $b \in \Sigma$. This graph has σ^r edges and every vertex of this graph has indegree and outdegree σ. Hence, it has a Eulerian cycle, which can be viewed as a sequence $u, b_1, b_2, \dots, b_{\sigma^r}$, where $u \in \Sigma^{r-1}$ is the start vertex, and the edge traversed in the i^{th} step is labelled with $b_i \in \Sigma$. Define $s_{\sigma,n}$ as the prefix of $ub_1b_2 \cdots b_{\sigma^r}$ of length n. The construction implies that $s_{\sigma,n}$ has $n - r + 1$ different substrings of length r. By the so-called mk-Lemma from [7], every SLP for $s_{\sigma,n}$ has size at least $\frac{n-r+1}{r} > \frac{n}{r} - 1 \geq \frac{n}{\log_\sigma(n)+1} - 1$. ◀

In [2] a set of n binary strings of length n is constructed such that any concatenation circuit that computes this set has size $\Omega(n^2/\log^2 n)$. A concatenation circuit for a set S of strings is simply an SLP such that every string from S is derived from a nonterminal of the SLP. Using the above construction, this lower bound can be improved to $\Omega(n^2/\log n)$: Simply take the string s_{2,n^2} and write it as $s_1 s_2 \cdots s_n$ with $|s_i| = n$. Then any concatenation circuit for $\{s_1, \dots, s_n\}$ has size $\Omega(n^2/\log n)$.

3 Trees and Tree Straight-Line Programs

For every $i \geq 0$, we fix a countably infinite set $\mathcal{F}_i$ (resp., $\mathcal{N}_i$) of *terminals* (resp., *nonterminals*) of rank i. Let $\mathcal{F} = \bigcup_{i \geq 0} \mathcal{F}_i$ and $\mathcal{N} = \bigcup_{i \geq 0} \mathcal{N}_i$. Moreover, let $\mathcal{X} = \{x_1, x_2, \dots\}$ be a countably infinite set of *parameters*. We assume that $\mathcal{F}$, $\mathcal{N}$, and $\mathcal{X}$ are pairwise disjoint. A *labelled tree* $t = (V, \lambda)$ is a finite, rooted and ordered tree t with node set V and labelling function $\lambda : V \to \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$. We require that a node $v \in V$ with $\lambda(v) \in \mathcal{F}_k \cup \mathcal{N}_k$ has exactly k children, which are ordered from left to right. We also require that every node v with $\lambda(v) \in \mathcal{X}$ is a leaf of t. The size of t is $|t| = |\{v \in V \mid \lambda(v) \in \mathcal{F} \cup \mathcal{N}\}|$, i. e., we do not count parameters. We denote trees in their usual term notation, e. g. $b(a, a)$ denotes the tree with a b-labelled root, which has two a-labelled children. We define $\mathcal{T}$ as the set of all labelled trees. The *depth* of a tree t is the maximal length (number of edges) of a path from the root to a leaf, and is denoted by $\mathrm{depth}(t)$. Let $\mathrm{labels}(t) = \{\lambda(v) \mid v \in V\}$ and $\mathcal{T}(\mathcal{L}) = \{t \mid \mathrm{labels}(t) \subseteq \mathcal{L}\}$ for $\mathcal{L} \subseteq \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$. We write $<_t$ for the depth-first-order on V. Formally, $u <_t v$ if u is an ancestor of v or if there exists a node w and $i < j$ such that the i^{th} child of w is an ancestor of u and the j^{th} child of w is an ancestor of v. The tree $t \in \mathcal{T}$ is *linear* if there do not exist different nodes that are labelled with the same parameter. We call $t \in \mathcal{T}$ *valid* if (i) $\mathrm{labels}(t) \cap \mathcal{X} = \{x_1, \dots, x_n\}$ for some $n \geq 0$ and (ii) for all $u, v \in V$ with $\lambda(u) = x_i$, $\lambda(v) = x_j$ and $u <_t v$ we have $i < j$ (in particular t is linear). For example, $f(x_1, x_{21}, x_{99})$, $f(x_1, x_1, x_3)$, and $f(x_3, x_1, x_2)$ are invalid, whereas $f(x_1, x_2, x_3)$ is valid. For a linear tree t we define $\mathrm{valid}(t)$ as the unique valid tree which is obtained from t by renaming the parameters. For instance, $\mathrm{valid}(f(x_{21}, x_2, x_{99})) = f(x_1, x_2, x_3)$. A valid tree t in which the parameters $x_1, \dots, x_n$ occur is also written as $t(x_1, \dots, x_n)$ and we write $\mathrm{rank}(t) = n$.

We now define a particular form of context-free tree grammars (see [8] for more details on context-free tree grammars) with the property that exactly one tree is derived. A *tree straight-line program (TSLP)* is a pair $\mathcal{G} = (S, P)$, where $S \in \mathcal{N}_0$ is the start nonterminal and P is a finite set of rules of the form $A(x_1, \dots, x_n) \to t(x_1, \dots, x_n)$ (which is also briefly written as $A \to t$), where $n \geq 0$, $A \in \mathcal{N}_n$ and $t(x_1, \dots, x_n) \in \mathcal{T}$ is valid such that:

- There is an initial rule $(S \to t) \in P$.

- If $(A \to s) \in P$ and $B \in \text{labels}(s) \cap \mathcal{N}$, then there is a tree t such that $(B \to t) \in P$.
- There do not exist rules $(A \to t_1), (A \to t_2) \in P$ with $t_1 \neq t_2$.
- The binary relation $\{(A, B) \in \mathcal{N} \times \mathcal{N} \mid (A \to t) \in P, B \in \text{labels}(t)\}$ is acyclic.

These conditions ensure that from every nonterminal $A \in \mathcal{N}_n$ exactly one valid tree $\text{val}_{\mathcal{G}}(A) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \dots, x_n\})$ is derived by using the rules as rewrite rules in the usual sense. The tree defined by $\mathcal{G}$ is $\text{val}(\mathcal{G}) = \text{val}_{\mathcal{G}}(S)$. Instead of a formal definition, we give an example:

▶ **Example 2.** Let $\mathcal{G} = (S, P)$, where P consists of the following rules ($a \in \mathcal{F}_0$, $b \in \mathcal{F}_2$): $S \to A(B)$, $A(x_1) \to C(F, x_1)$, $B \to E(F)$, $C(x_1, x_2) \to D(E(x_1), x_2)$, $D(x_1, x_2) \to b(x_1, x_2)$, $E(x_1) \to D(F, x_1)$, $F \to a$. Part of a possible derivation of $\text{val}(\mathcal{G}) = b(b(a, a), b(a, a))$ from S is: $S \to A(B) \to C(F, B) \to D(E(F), B) \to b(E(F), B) \to b(D(F, F), B) \to b(b(F, F), B) \to b(b(a, F), B) \to b(b(a, a), B) \to b(b(a, a), E(F)) \to \cdots \to b(b(a, a), b(a, a))$.

The size $|\mathcal{G}|$ of a TSLP $\mathcal{G} = (S, P)$ is the total size of all trees on the right-hand sides of P: $|\mathcal{G}| = \sum_{(A \to t) \in P} |t|$. For instance, the TSLP from Example 2 has size 12.

A TSLP is in *Chomsky normal form* if for every production $A(x_1, \dots, x_n) \to t(x_1, \dots, x_n)$ one of the following two cases holds:

$$t(x_1, \dots, x_n) = B(x_1, \dots, x_{i-1}, C(x_i, \dots, x_k), x_{k+1}, \dots, x_n) \text{ for } B, C \in \mathcal{N} \tag{1}$$

$$t(x_1, \dots, x_n) = f(x_1, \dots, x_n) \text{ for } f \in \mathcal{F}_n. \tag{2}$$

If the tree t in the corresponding rule $A \to t$ is of type (1), we write $\text{index}(A) = i$. If otherwise t is of type (2), we write $\text{index}(A) = 0$. One can transform every TSLP efficiently into an equivalent TSLP in Chomsky normal form with a small size increase [24]. We only consider TSLPs in Chomsky normal form in the following.

We define the rooted, ordered derivation tree $\mathcal{D}_{\mathcal{G}}$ of a TSLP $\mathcal{G} = (S, P)$ in Chomsky normal form as for string grammars: The inner nodes of the derivation tree are labelled by nonterminals and the leaves are labelled by terminal symbols. Formally, we start with the root node of $\mathcal{D}_{\mathcal{G}}$ and assign it the label S. For every node in $D_{\mathcal{G}}$ labelled by A, where the right-hand side t of the rule for A is of type (1), we attach a left child labelled by B and a right child labelled by C. If the right-hand side t of the rule for A is of type (2), we attach a single child labelled by f to A. Note that these nodes are the leaves of $\mathcal{D}_{\mathcal{G}}$ and they represent the nodes of the initial tree $\text{val}(\mathcal{G})$. We denote by $\text{depth}(\mathcal{G})$ the depth of the derivation tree $\mathcal{D}_{\mathcal{G}}$. For instance, the depth of the TSLP from Example 2 is 4.

A commonly used compact tree compression scheme is obtained by writing down repeated subtrees only once. In that case all occurrences except for the first are replaced by a pointer to the first one. This leads to a node-labelled *directed acyclic graph* (dag). It is known that every tree has a unique minimal dag, which is called the *the dag* of the initial tree. An example can be found in Figure 2, where the right graph is the dag of the tree in the middle. The dag of a tree t can be constructed in time $O(|t|)$ [10]. Dags correspond to TSLPs where every nonterminal has rank 0.

4 Constructing a small TSLP for a tree

In this section we construct a TSLP $\mathcal{G}$ for a given tree t of size n. We then prove that $|\mathcal{G}| \in O(n/\log n)$. For the remainder of this section we restrict our input to binary trees, i. e., every node has either zero or two children. Formally, we consider trees from $\mathcal{T}(\mathcal{F}_0 \cup \mathcal{F}_2)$.

The following idea of splitting a tree recursively into smaller parts of roughly equal size is well-known, see e. g. [4, 26]. For our later analysis, it is important to bound the number of parameters in the resulting nonterminals (i. e., the number of holes in trees)

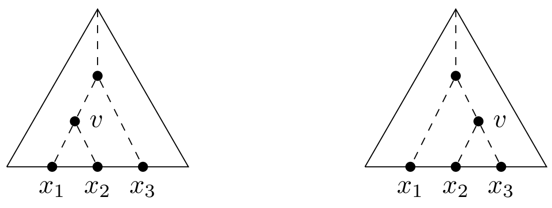


Figure 1 Splitting a tree with three parameters.

by a constant. To achieve this, we use an idea from Ruzzo's paper [25]. For a valid tree $t = (V, \lambda) \in \mathcal{T}(\mathcal{F}_0 \cup \mathcal{F}_2 \cup \mathcal{X})$ and a node $v \in V$ we denote by $t[v]$ the tree valid(s), where s is the subtree rooted at v in t. We further write $t \setminus v$ for the tree valid(r), where r is obtained from t by replacing the subtree rooted at v by a new parameter. If for instance $t = h(g(x_1, f(x_2, x_3)), x_4)$ and v is the f-labelled node, then $t[v] = f(x_1, x_2)$ and $t \setminus v = h(g(x_1, x_2), x_3)$. The following lemma is well-known, see e. g. [19].

▶ **Lemma 3.** *Let t be a binary tree with $|t| \geq 2$. One can determine in time $O(|t|)$ a node v such that $\frac{1}{3}|t| - \frac{1}{2} \leq |t[v]| \leq \frac{2}{3}|t|$.*

For the remainder of this section we denote by split(t) the unique node in a tree t computed using Lemma 3. We now construct a TSLP $\mathcal{G}$ with val($\mathcal{G}$) $= t$ for a given binary tree t (we assume that $|t| \geq 2$). Every nonterminal of $\mathcal{G}$ will be of rank at most three. We store two sets of productions, P_{temp} and P_{final}. The set P_{final} contains rules of the final TSLP $\mathcal{G}$ and P_{temp} ensures that the TSLP $(S, P_{\text{temp}} \cup P_{\text{final}})$ produces t at any point of time. Initially, we set $P_{\text{temp}} := \{S \to t\}$ and $P_{\text{final}} := \emptyset$. While P_{temp} is non-empty we proceed for each rule $(A \to s) \in P_{\text{temp}}$ as follows: Let $A \in \mathcal{N}_r$. If $r \leq 2$ we determine the node $v = \text{split}(s)$ in s. Then we split the tree s into the trees $s[v]$ and $s \setminus v$. Let $r_1 = \text{rank}(s[v])$, $r_2 = \text{rank}(s \setminus v)$ and let $A_1 \in \mathcal{N}_{r_1}$ and $A_2 \in \mathcal{N}_{r_2}$ be fresh nonterminals. Note that $r = r_1 + r_2 - 1$. If the size of $s[v]$ (resp., $s \setminus v$) is larger than 1 we add the rule $A_1 \to s[v]$ (resp., $A_2 \to s \setminus v$) to P_{temp}. Otherwise we add it to P_{final} as a final rule. Let k be the number of nodes of s that are labelled by a parameter and that are smaller (w.r.t. $<_s$) than v. To link the nonterminal A to the fresh nonterminals A_1 and A_2 we add the rule $A(x_1, \ldots, x_r) \to A_1(x_1, \ldots, x_k, A_2(x_{k+1}, \ldots, x_{k+r_2}), x_{k+r_2+1}, \ldots, x_r)$ to P_{final}.

To bound the rank of the nonterminals by three we handle rules $A \to s$ with $A \in \mathcal{N}_3$ as follows. Let v_1, v_2 and v_3 be the nodes labelled by the parameters x_1, x_2 and x_3, respectively. Instead of choosing the node v by split(s) we set v to the lowest common ancestor of (v_1, v_2) or (v_2, v_3), depending on which one has the greater distance from the root node (see Figure 1). This step ensures that the two trees $s[v]$ and $s \setminus v$ have rank 2, so in the next step each of them will be split in a balanced way according to Lemma 3. As a consequence, the resulting TSLP has depth $O(\log |t|)$ but size $O(|t|)$. The running time of this first phase can be bounded by $O(|t| \log |t|)$: All right-hand sides from P_{temp} obtained after i splittings have total size at most $|t|$, so we need time $O(|t|)$ to split them. Moreover, i ranges from 0 to $O(\log |t|)$.

▶ **Example 4.** If we apply our construction to the tree $b(b(a, a), b(a, a))$ we get the TSLP with the rules $S \to A(B)$, $A(x_1) \to C(D, x_1)$, $B \to E(F)$, $C(x_1, x_2) \to G(H(x_1), x_2)$, $D \to a$, $E(x_1) \to I(J, x_1)$, $F \to a$, $G(x_1, x_2) \to b(x_1, x_2)$, $H(x_1) \to K(L(x_1))$, $I(x_1, x_2) \to b(x_1, x_2)$, $J \to a$, $K(x_1, x_2) \to b(x_1, x_2)$, and $L \to a$.

In the next step we want to compact the TSLP by considering the dag of the derivation tree. For this we first build the derivation tree $\mathcal{D}_{\mathcal{G}}$ from the TSLP $\mathcal{G}$ as described above. The derivation tree for the TSLP described in Example 4 is shown on the left of Figure 2.

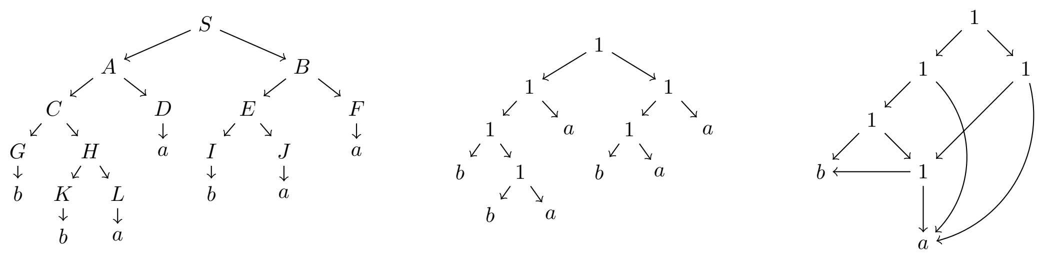


Figure 2 The derivation tree from Example 4.

We now want to identify some (but not all) nonterminals that produce the same tree. Note that if we just omit the nonterminal labels from the derivation tree, then there might exist isomorphic subtrees of the derivation whose root nonterminals produce different trees. This is due to the fact that we lost for an A-labelled node of the derivation tree with a left (resp., right) child that is labelled with B (resp., C) the information at which argument position of B the nonterminal C is substituted. To keep this information we replace every label A in the derivation tree with $\mathrm{index}(A) \in \{0,1,2,3\}$ (the index of a nonterminal of a TSLP in Chomsky normal form was defined in Section 3). Moreover, we remove every leaf v and write its label into its parent node. We call the resulting tree the *modified derivation tree* and denote it by $\mathcal{D}^*_{\mathcal{G}}$. Note that $\mathcal{D}^*_{\mathcal{G}}$ is a full binary tree with node labels from $\{1,2,3\} \cup \mathrm{labels}(t)$. The modified derivation tree for Example 4 is shown in the middle of Figure 2. The following lemma shows how to compact our grammar by considering the dag of $\mathcal{D}^*_{\mathcal{G}}$.

▶ **Lemma 5.** *Let u and v be nodes of $\mathcal{D}_{\mathcal{G}}$ labelled by A resp. B. Moreover, let u' and v' be the corresponding nodes in $\mathcal{D}^*_{\mathcal{G}}$. If the subtrees $\mathcal{D}^*_{\mathcal{G}}[u']$ and $\mathcal{D}^*_{\mathcal{G}}[v']$ are isomorphic (as labelled ordered trees), then $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}}(B)$.*

By Lemma 5, if two subtrees of $\mathcal{D}^*_{\mathcal{G}}$ are isomorphic we can eliminate the nonterminal of a root node of one subtree. Hence, we construct the dag d of $\mathcal{D}^*_{\mathcal{G}}$. This is possible in time $O(|\mathcal{D}_{\mathcal{G}}|) = O(|t|)$ [10]. The minimal dag of the TSLP of Example 4 is shown on the right of Figure 2. The nodes of d are the nonterminals of the final TSLP. We obtain rules of type (1) for each nonterminal corresponding to an inner node of d and rules of type (2) for each leaf in d. Let n_1 be the number of inner nodes of d and n_2 be the number of leaves. Then the size of our final TSLP is $2n_1 + n_2$, which is bounded by twice the number of nodes of d. The dag from Figure 2 gives the TSLP for the tree $b(b(a,a), b(a,a))$ described in Example 2.

To estimate the number of nodes in the dag of the modified derivation tree, we prove in this section a general result about the size of dags of certain weakly balanced binary trees. Let t be a binary tree and let $0 < \beta < 1$ and $\gamma \geq 2$ be constants. The *leaf size* of a node v is the number of leaves of the subtree rooted at v. We say that an inner node v with children v_1 and v_2 is *β-balanced* if the following holds: If n_i is the leaf size of v_i, then $n_1 \geq \beta n_2$ and $n_2 \geq \beta n_1$. We say that t is *(β,γ)-balanced* if the following holds: For all inner nodes u and v such that v is a child of u and the leaf size of v (and hence also u) is at least γ, we have that u is β-balanced or v is β-balanced.

▶ **Theorem 6.** *Fix constants $0 < \beta < 1$ and $\gamma \geq 2$. Then there is a constant α (depending on β and γ) such that the following holds: If t is a (β,γ)-balanced binary tree with n leaves and $|\mathrm{labels}(t)| = \sigma$ (hence, $|t|, \sigma \leq 2n-1$), then the size of the dag of t is bounded by $\frac{\alpha \cdot n}{\log_\sigma n}$.*

Proof. Let us fix a tree $t = (V,\lambda)$ as in the theorem with n leaves. Moreover, let us fix a number $k \geq \gamma$ that will be defined later. Let $\mathrm{top}(t,k)$ be the tree obtained from t by

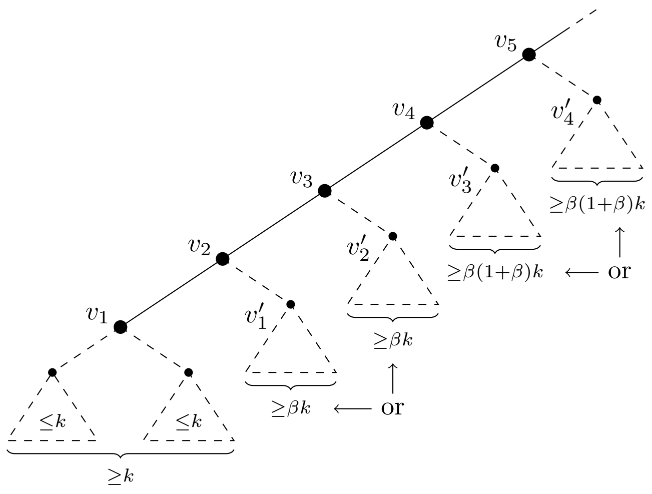


Figure 3 A chain within a top tree. The subtree rooted at v_1 has more than k leaves.

removing all nodes with leaf size at most k. We first bound the number of different subtrees with at most k leaves in t. Afterwards we will estimate the size of the remaining tree $\text{top}(t, k)$. The same strategy is used for instance in [13, 20] to derive a worst-case upper bound on the size of binary decision diagrams.

Claim 1. The number of different subtrees of t with at most k leaves is bounded by d^k with $d = 4\sigma^2$.

A subtree of t with i leaves has exactly $2i - 1$ nodes, each labelled with one of σ labels. Let $C_m = \frac{1}{m+1}\binom{2m}{m}$ be the mth Catalan number. It is known that $C_m \leq 4^m$. If the labels are ignored, there are C_{i-1} different subtrees with i leaves. In conclusion, we get the following bound: $\sum_{i=1}^{k} C_{i-1} \cdot \sigma^{2i-1} \leq \sum_{i=0}^{k-1} 4^i \cdot \sigma^{2i+1} = \sigma \frac{(4\sigma^2)^k - 1}{4\sigma^2 - 1} \leq (4\sigma^2)^k$.

Claim 2. The number of nodes of $\text{top}(t, k)$ is bounded by $c \cdot \frac{n}{k}$ for a constant c depending only on β and γ.

The tree $\text{top}(t, k)$ has at most n/k leaves since it is obtained from t by removing all nodes with leaf size at most k. Each node in $\text{top}(t, k)$ has at most two children. Therefore it remains to show that the length of *unary chains* in $\text{top}(t, k)$ is bounded by a constant.

Let $v_1, \ldots, v_m$ be a unary chain in $\text{top}(t, k)$ where v_i is the single child node of v_{i+1}. Moreover, let v'_i be the removed sibling of v_i in t, see Figure 3. Note that each node v'_i has leaf size at most k. We claim that the leaf size of v_{2i+1} is larger than $(1 + \beta)^i k$ for all i with $2i + 1 \leq m$. For $i = 0$ note that v_1 has leaf size more than k since otherwise it would have been removed in $\text{top}(t, k)$. For the induction step, assume that the leaf size of v_{2i-1} is larger than $(1 + \beta)^{i-1} k \geq k \geq \gamma$. One of the nodes v_{2i} and v_{2i+1} must be β-balanced. Hence, v'_{2i-1} or v'_{2i} must have leaf size more than $\beta(1 + \beta)^{i-1} k$. Hence, v_{2i+1} has leaf size more than $(1 + \beta)^{i-1} k + \beta(1 + \beta)^{i-1} k = (1 + \beta)^i k$.

Let $\ell = \log_{1+\beta}(\beta^{-1})$. If $m \geq 2\ell + 3$, then $v_{2\ell+1}$ exists and has leaf size more than k/β, which implies that the leaf size of $v'_{2\ell+1}$ or $v'_{2\ell+2}$ (both nodes exist) is more than k, which is a contradiction. Hence, we must have $m \leq 2 \log_{1+\beta}(\beta^{-1}) + 2$. Figure 3 shows an illustration.

Using Claim 1 and 2 we can now prove the theorem: The number of nodes of the dag of t is bounded by the number of different subtrees with at most k leaves (Claim 1) plus the number of nodes of the remaining tree $\text{top}(t, k)$ (Claim 2). Let $k = \max\{\gamma, \frac{1}{2} \log_d n\} \geq \gamma$

(recall that $d = 4\sigma^2$ and hence $\log d = 2 + 2\log\sigma$). If $k = \gamma$, i. e., $\frac{1}{2}\log_d n \le \gamma$ then we have $\frac{n}{\log_\sigma n} \in \Omega(n)$ and the bound $O(\frac{n}{\log_\sigma n})$ on the size of the dag is trivial. If $k = \frac{1}{2}\log_d n$ then we get with Claim 1 and 2 the following bound on the size of the dag: $d^k + c \cdot \frac{n}{k} = d^{(\log_d n)/2} + 2c \cdot \frac{n}{\log_d n} = \sqrt{n} + 2c \cdot \frac{n}{\log_d n} \in O(\frac{n}{\log_d n}) = O(\frac{n}{\log_\sigma n})$. This proves the theorem. ◀

Obviously, one could relax the definition of (β,γ)-balanced by only requiring that if $(v_1, v_2, \dots, v_\delta)$ is a path down in the tree, where δ is a constant and v_δ has leaf size at least γ, then one of the nodes $v_1, v_2, \dots, v_\delta$ must be β-balanced. Theorem 6 would still hold with this definition (with the constant α also depending on δ).

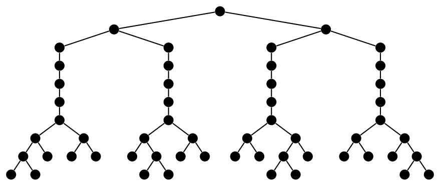
Let us fix the TSLP $\mathcal{G}$ for a binary tree $t \in \mathcal{T}(\mathcal{F}_0 \cup \mathcal{F}_2)$ that has been produced by the first part of our algorithm. Let $n = |t|$ and $\sigma = |\text{labels}(t)|$. Then, the modified derivation tree $\mathcal{D}^*_\mathcal{G}$ is a binary tree with n leaves (and hence $2n-1$ nodes) and $\sigma + 3$ different node labels (namely $1, 2, 3$ and those appearing in t). Moreover, $\mathcal{D}^*_\mathcal{G}$ is $(1/3, 6)$-balanced: If we have two successive nodes in $\mathcal{D}^*_\mathcal{G}$, then we split at one of the two nodes according to Lemma 3. Now, assume that we split at node v according to Lemma 3. Let v_1 and v_2 be the children of v, let n_i be the leaf size of v_i, and let $n = n_1 + n_2 \ge 6$ be the leaf size of v. We get $\frac{1}{3}n - \frac{1}{2} \le n_1 \le \frac{2}{3}n$ and $\frac{1}{3}n \le n_2 \le \frac{2}{3}n + \frac{1}{2}$ (or vice versa). Since $n \ge 6$ we have $\frac{1}{4}n \le n_1 \le \frac{2}{3}n$ and $\frac{1}{3}n \le n_2 \le \frac{3}{4}n$. We get $n_1 \ge \frac{1}{4}n \ge \frac{1}{3}n_2$ and $n_2 \ge \frac{1}{3}n \ge \frac{1}{2}n_1$. Hence, we get:

▶ **Corollary 7.** *Let t be a binary tree with $|t| = n$ and $|\text{labels}(t)| = \sigma$. Let d be the minimal dag of the modified derivation tree produced from t by our algorithm. Then the number of nodes of d is in $O\big(\frac{n}{\log_\sigma n}\big)$. Hence, the size of the TSLP produced from t is in $O\big(\frac{n}{\log_\sigma n}\big)$.*

The conditions in Theorem 6 ensure that $\text{depth}(t) \in O(\log|t|)$. One might think that a tree t of depth $O(\log|t|)$ has a small dag. For instance, the dag of a complete binary tree with n nodes has size $O(\log n)$. But this intuition is wrong:

▶ **Theorem 8.** *There is a family of trees $t_n \in \mathcal{T}(\{a, c\})$ ($a \in \mathcal{F}_0$, $c \in \mathcal{F}_2$), $n \ge 1$, such that* (i) $|t_n| \in O(n)$, (ii) $\text{depth}(t) \in O(\log n)$, *and* (iii) *the size of the dag of t_n is at least n.*

Proof. To simplify the presentation, we use a unary node label $b \in \mathcal{F}_1$. It can be replaced by the pattern $c(d, x)$, where $d \in \mathcal{F}_0 \setminus \{a\}$ to obtain a binary tree. Let $k = \frac{n}{\log n}$ (we ignore rounding problems with $\log n$, which only affects multiplicative factors). Choose k different binary trees $s_1, \dots, s_k \in \mathcal{T}(\{a, c\})$, each having $\log n$ internal nodes. Note that this is possible since by the formula for the Catalan numbers there are more than n different binary trees with $\log n$ internal nodes for n large enough. Then consider the trees $s'_i = b^{\log n}(s_i)$. Each of these trees has size at most $3\log n$ as well as depth at most $3\log n$. Next, let $u_n(x_1, \dots, x_k) \in \mathcal{T}(\{c, x_1, \dots, x_k\})$ a binary tree (all non-parameter nodes are labelled with c) of depth $\log k \le \log n$ and size $O(k) = O(\frac{n}{\log n})$. We finally take $t_n = u_n(s'_1, \dots, s'_k)$. A possible choice for t_{16} is shown below. We obtain $|t_n| = O(\frac{n}{\log n}) + O(k \cdot \log n) = O(n)$. The depth of t_n is bounded by $3\log n$. Finally, in the dag for t_n the unary b-labelled nodes cannot be shared. Basically, the pairwise different trees $t_1, \dots, t_n$ work as different constants that are attached to the b-chains. But the number of b-labelled nodes in t_n is $k \cdot \log n = n$. ◀



It is straightforward to adapt our algorithm to trees where every node has at most r children for a fixed constant r. One only has to prove a version of Lemma 3 for r-ary trees. The multiplicative constant in the $O\big(\frac{n}{\log_\sigma n}\big)$ bound for the final TSLP will depend on r. On the other hand, for unranked trees, where the number of children of a node is arbitrary, our algorithm does not work. This problem can be solved by transforming an unranked tree into a binary tree of the same size using the first-child next-sibling encoding [18]. For this binary tree we get a TSLP of size $O\big(\frac{n}{\log_\sigma n}\big)$.

For traversing a compressed unranked tree t, another well-known encoding is favorable. Let c_t be a compressed representation (e. g., a TSLP) of t. The goal is to represent t in space $O(|c_t|)$ such that one can efficiently navigate from a node to (i) its parent node, (ii) its first child, (iii) its next sibling, and (iv) its previous sibling (if they exist). For top dags [3], it was shown that a single navigation step can be done in time $O(\log |t|)$. Using the right binary encoding, we can prove the same result for TSLPs: Let r be the maximal rank of a node of the unranked tree t. We define the binary encoding $\mathrm{bin}(t)$ by adding for every node v of rank $s \le r$ a binary tree of depth $\lceil \log s \rceil$ with s many leaves, whose root is v and whose leaves are the children of v. This introduces at most $2s$ many new binary nodes, which are labelled by a new symbol. We get $|\mathrm{bin}(t)| \le 3|t|$. In particular, we obtain a TSLP of size $O\big(\frac{n}{\log_\sigma n}\big)$ for $\mathrm{bin}(t)$, where $n = |t|$ and $\sigma = |\mathrm{labels}(t)|$. Note that a traversal step in the initial tree t (going to the parent node, first child, next sibling, or previous sibling) can be simulated by $O(\log r)$ many traversal steps in $\mathrm{bin}(t)$ (going to the parent node, left child, or right child). But for a binary tree s, it was recently shown that a TSLP $\mathcal{G}$ for s can be represented in space $O(|\mathcal{G}|)$ such that a single traversal step takes time $O(1)$ [21] (this generalizes a corresponding result for strings [12]). Hence, we can navigate in t in time $O(\log r) \le O(\log |t|)$.

5 Arithmetical Circuits

In this section, we present our main application of Theorem 7. Let $\mathcal{S} = (S, +, \cdot)$ be a (not necessarily commutative) semiring. Thus, $(S, +)$ is a commutative monoid with identity element 0, $(S, \cdot)$ is a monoid with identity element 1, and $\cdot$ left and right distributes over $+$. We use the standard notation of arithmetical formulas and circuits over $\mathcal{S}$: An *arithmetical formula* (resp. *arithmetical circuit*) is a binary tree (resp. dag) where internal nodes are labelled with the semiring operations $+$ and $\cdot$, and leaf nodes are labelled with variables $y_1, y_2, \dots$ or the constants 0 and 1. The *depth* of a circuit is the length of a longest path from the root node to a leaf. An arithmetical circuit evaluates to a multivariate noncommutative polynomial $p(y_1, \dots, y_n)$ over $\mathcal{S}$, where $y_1, \dots, y_n$ are the variables occurring at the leaf nodes. Two arithmetical circuits are equivalent if they evaluate to the same polynomial. Brent [4] has shown that every arithmetical formula of size n over a commutative ring can be transformed into an equivalent circuit of depth $O(\log n)$ and size $O(n)$ (the proof easily generalizes to semirings). Using Theorem 7 we can refine the size bound to $O(\frac{n \cdot \log m}{\log n})$, where m is the number of different variables in the formula:

▶ **Theorem 9.** *An arithmetical formula F of size n with m different variables can be transformed in time $O(n \log n)$ into an arithmetical circuit C of depth $O(\log n)$ and size $O(\frac{n \cdot \log m}{\log n})$ such that C and F are equivalent for every semiring.*

Proof sketch. Fix a semiring $\mathcal{S}$. We apply our TSLP construction to the formula tree F and obtain a TSLP $\mathcal{G}$ for F of size $O(\frac{n \cdot \log m}{\log n})$ and depth $O(\log n)$. Using the main construction from [24] we can reduce the rank of nonterminals in $\mathcal{G}$ to 1. Thereby the size and depth of the TSLP only increase by constant factors. Recall that for a nonterminal $A(x)$, $\mathrm{val}_{\mathcal{G}}(A)$ is a tree in which each of the parameter x occurs exactly once. By evaluating this tree in the polynomial semiring $\mathcal{S}[y_1, \dots, y_m]$, we obtain a noncommutative polynomial $p_A(x) = a_0 + a_1 x a_2$, where $a_0, a_1, a_2 \in \mathcal{S}[y_1, \dots, y_m]$. We now transform $\mathcal{G}$ into an arithmetical circuit that contains for every nonterminal A of rank one gates that evaluate to the above polynomials a_0, a_1, a_2. E.g., for a rule of the form $A(x) \to B(C(x))$ one has to substitute the polynomial $p_C(x)$ into $p_B(x)$ and carry out the obvious simplifications. For a nonterminal A of rank 0, the circuit simply contains a gate that evaluates to the polynomial to which $\mathrm{val}_{\mathcal{G}}(A)$ evaluates. ◀

Theorem 9 can also be shown for fields instead of semirings. In this case, the expression is built up using variables, the constants -1, 0, 1, and the field operations $+, \cdot$ and $/$.

6 Future work

In [27] a universal (in the information-theoretic sense) code for binary trees is developed. This code is computed in two phases: In a first step, the minimal dag for the input tree is constructed. Then, a particular binary encoding is applied to the dag. It is shown that the *average redundancy* of the resulting code converges to zero (see [27] for definitions) for every probability distribution on binary trees that satisfies the so-called domination property (a somewhat technical condition) and the representation ratio negligibility property. The latter means that the average size of the dag divided by the tree size converges to zero for the underlying probability distribution. This is, for instance, the case for the uniform distribution, since the average size of the dag is $\Theta(n/\sqrt{\log n})$ [11]. We are confident that replacing the minimal dag by a TSLP of size $O(\frac{n}{\log n})$ in the universal tree encoder from [27] leads to stronger results. In particular, we hope to get a code whose *maximal pointwise redundancy* converges to zero for certain probability distributions. For strings, such a result was obtained in [16] using the fact that every string of length n has an SLP of size $O(\frac{n}{\log n})$.

Another interesting question is whether the time bound of $O(n \log n)$ for the construction of a TSLP of size $O(\frac{n}{\log_\sigma n})$ can be improved to $O(n)$. Related to this is the question for the worst-case output size of the grammar-based tree compressor from [15]. It works in linear time and produces a TSLP that is only by a factor $O(\log n)$ larger than an optimal TSLP. From a complexity theoretic point of view, it would be also interesting to see, whether our TSLP construction can be carried out in logarithmic space or even NC^1.

In [3] the authors proved that the top dag of a given tree t of size n is at most by a factor $\log n$ larger than the minimal dag of t. It is not clear, whether the TSLP constructed by our algorithm has this property too. The construction of the top dag is done in a bottom-up way, and as a consequence identical subtrees are compressed in the same way. This property is crucial for the comparison with the minimal dag. Our algorithm works in a top-down way. Hence, it is not guaranteed that identical subtrees are compressed in the same way.

Finally, one should also study whether all the operations from [3] for top dags can be implemented with the same time bounds also for TSLPs. For traversing the tree, this is possible, see the paragraph at the end of Section 4. For the other operations, like for instance computing lowest common ancestors, this is not clear.

Acknowledgment. We have to thank Anna Gál for helpful comments.

References

1 T. Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Inf. Process. Lett.*, 110(18-19):815–820, 2010.

2 V. Arvind, S. Raja, and A. V. Sreejith. On lower bounds for multiplicative circuits and linear circuits in noncommutative domains. In *Proc. CSR 2014*, volume 8476 of *LNCS*, pages 65–76. Springer, 2014.

3 P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. In *Proc. ICALP (1) 2013*, volume 7965 of *LNCS*, pages 160–171. Springer, 2013.

4 R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

5 N. H. Bshouty, R. Cleve, and W. Eberly. Size-depth tradeoffs for algebraic formulas. *SIAM J. Comput.*, 24(4):682–705, 1995.

6 G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4–5):456–474, 2008.

7 M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.

8 H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://tata.gforge.inria.fr`.

9 N. de Bruijn. A combinatorial problem. *Nederl. Akad. Wet., Proc.*, 49:758–764, 1946.

10 P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

11 P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Proc. ICALP 1990*, volume 443 of *LNCS*, pages 220–234. Springer, 1990.

12 L. Gasieniec, R. M. Kolpakov, I. Potapov, and P. Sant. Real-time traversal in grammar-based compressed files. In *Proc. DCC 2005*, page 458. IEEE Computer Society, 2005.

13 M. A. Heap and M. R. Mercer. Least upper bounds on OBDD sizes. *IEEE Trans. Computers*, 43(6):764–767, 1994.

14 D. Hucke, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. arXiv.org, `http://arxiv.org/abs/1407.4286`, 2014.

15 A. Jeż and M. Lohrey. Approximation of smallest linear tree grammars. In *Proc. STACS 2014*, volume 25 of *LIPIcs*, pages 445–457. Leibniz-Zentrum für Informatik, 2014.

16 J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.

17 J. C. Kieffer, E.-H. Yang, G. J. Nelson, and P. C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans.Inf. Theory*, 46(4):1227–1245, 2000.

18 D. E. Knuth. *The Art of Computer Programming, Vol. I: Fundamental Algorithms.* Addison-Wesley, 1968.

19 P. M. Lewis II, R. E. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proc. 6th Annual IEEE Symp. Switching Circuit Theory and Logic Design*, pages 191–202, 1965.

20 H.-T. Liaw and C.-S. Lin. On the OBDD-representation of general boolean functions. *IEEE Trans. Computers*, 41(6):661–664, 1992.

21 M. Lohrey. Traversing grammar-compressed trees with constant delay. `http://www.eti.uni-siegen.de/ti/veroeffentlichungen/14-traversal.pdf`.

22 M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

23 M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Inf. Syst.*, 38(8):1150–1167, 2013.

24 M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.

25 W. L. Ruzzo. Tree–size bounded alternation. *J. Comput. Syst. Sci.*, 21:218–235, 1980.

26 P. M. Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proc. 4th Hawaii Symp. on System Sciences*, pages 525–527, 1971.

27 J. Zhang, E.-H. Yang, and J. C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Trans. Inf. Theory*, 60(3):1373–1386, 2014.

One Time-traveling Bit is as Good as Logarithmically Many*

Ryan O'Donnell[1] **and A. C. Cem Say**[2]

1 **Computer Science Department, Carnegie Mellon University Pittsburgh, USA**
`odonnell@cs.cmu.edu`

2 **Department of Computer Engineering, Boğaziçi University İstanbul, Turkey**
`say@boun.edu.tr`

Abstract

We consider computation in the presence of closed timelike curves (CTCs), as proposed by Deutsch. We focus on the case in which the CTCs carry *classical* bits (as opposed to qubits). Previously, Aaronson and Watrous showed that computation with polynomially many CTC bits is equivalent in power to PSPACE. On the other hand, Say and Yakaryılmaz showed that computation with just 1 classical CTC bit gives the power of "postselection", thereby upgrading classical randomized computation (BPP) to the complexity class $\mathsf{BPP}_{\mathsf{path}}$ and standard quantum computation (BQP) to the complexity class PP. It is natural to ask whether increasing the number of CTC bits from 1 to 2 (or 3, 4, etc.) leads to increased computational power. We show that the answer is no: randomized computation with logarithmically many CTC bits (i.e., polynomially many CTC states) is equivalent to $\mathsf{BPP}_{\mathsf{path}}$. (Similarly, quantum computation augmented with logarithmically many *classical* CTC bits is equivalent to PP.) Spoilsports with no interest in time travel may view our results as concerning the robustness of the class $\mathsf{BPP}_{\mathsf{path}}$ and the computational complexity of sampling from an implicitly defined Markov chain.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases Time travel, postselection, Markov chains, randomized computation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.469

1 On time travel

We begin with a discussion of time travel. Readers not interested in this concept may skip directly to Section 2, wherein we define the problem under consideration in a purely complexity-theoretic manner, with no reference to time travel.

Kurt Gödel [20] was the first to point out that Einstein's theory of general relativity is consistent with the existence of *closed timelike curves* (CTCs), raising the theoretical possibility of time travel. Any model of time travel must deal with the "Grandfather Paradox", wherein a trip to the past causes a chain of events that leads to a future in which that very trip does not take place. Assume that a time-traveler changes the state of the universe at the earlier end t_0 of a time loop from state s to some different state s'. Then just what is the state of the universe at time t_0: is it s or s'? Seeing a logical inconsistency in this scenario,

* Ryan O'Donnell's work performed while at the Boğaziçi University Computer Engineering Department, supported by Marie Curie International Incoming Fellowship project number 626373.

most thinkers of earlier generations concluded that time travel to the past must be impossible. There is, however, a way out. In an influential paper [18], Friedman *et al.* suggested Nature might allow CTCs as long as they do not “change the past”, an idea that has come to be known as the *Novikov self-consistency principle.* The main two rivaling models of time travel – the “Deutschian model” (which we study in this work), and the “postselected CTC model” from [26] – both conform to the Novikov self-consistency principle.

In the model put forward by Deutsch [16], the universe need not be in a single deterministic state at time t_0. Rather, the state of the universe should be viewed as a probability distribution over several states (possibly even quantum states) like s and s' in the example above. The requirement that the past should not change is fulfilled by stipulating that Nature sets the state x of the portion of the universe affected by the CTC at time t_0 to a fixed point of the operator f describing the evolution in the CTC (meaning $x = f(x)$).

To take the traditional example, suppose a deranged scientist can access a CTC to the past century, and he sends through it a bomb that is programmed to kill his grandfather (who is only a child back then). We consider two (classical) states of the universe at the time of the bomb's arrival: state 1 is “grandfather dies” and state 2 is “grandfather lives”. We assume the universe proceeds deterministically from that point on: if the grandfather is killed, then in the future no bomb is sent through the CTC; conversely, if the grandfather lives, then the deranged scientist is born and *does* send the bomb back in time. We can model this evolution by a 2-state Markov chain with the following transition matrix (that happens to be deterministic): $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. In Deutsch's model, Nature sets the state of the universe to be the stationary distribution for this chain: $\begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$. That is, the bomb arrives to kill the grandfather with probability $\frac{1}{2}$.

1.1 Computation with CTCs

As the reader can see, Nature performs a kind of computation here, determining the stationary distribution of the Markov chain that has been arranged within the CTC by the deranged scientist. It is natural to wonder if Nature's power can be effectively harnessed by a computational device. Indeed, Deutsch [16] pointed out that in general his model involves Nature solving an NP-hard problem; later, Brun [15] discussed the possibility of using CTCs to solve the Factoring problem efficiently. The first clear model of computation with Deutschian CTCs was proposed by Bacon [11]. Both Deutsch and Bacon consider sending *qubits* through a CTC. However, as pointed out by Aaronson [3], it is also very interesting (and simpler) to consider only classical bits passing through a CTC. Indeed, as far as we aware, there are no results showing that time-traveling quantum bits confer a computational advantage over time-traveling classical bits. Therefore, in the rest of this section we will sketch the Deutschian model of computation with classical CTC bits, and mention prior work. A formal complexity-theoretic definition of the model (with no reference to time travel) is given in Section 2.

Suppose that a computational agent $\mathcal{A}$ has access to a CTC which is “wide” enough to support the transmission of w bits. Thus the physical object being sent through the CTC can be in one of $S = 2^w$ states. (We may also more generally consider values of S that are not powers of 2.) Let us think of $\mathcal{A}$ as a classical polynomial-time randomized Turing machine (though it might be of another type; e.g., a BQP-machine). Say that $\mathcal{A}$ is trying to decide if a given input $x \in \{0,1\}^n$ belongs to language L. The algorithm $\mathcal{A}$ can read the w bits in the CTC, perform some computation, and then send a new string of w bits through the CTC. Since the incoming and outgoing bit strings can be in one of $2^w = S$ states, and since $\mathcal{A}$ is a randomized algorithm, the operation of $\mathcal{A}$ on the CTC constitutes an S-state Markov chain M_x, which depends on the input $x \in \{0,1\}^n$ to the L-decision problem.

In the Deutschian model, we assume that once the Markov chain M_x is defined, Nature sets the distribution of the bits in the CTC to *some* stationary distribution of M_x. We emphasize that it's merely *some* stationary distribution (at least one of which always exists) – we don't assume that M_x must have a unique stationary distribution. (Now is a good time to mention that if $\mathcal{A}$ is allowed to send qubits along the CTC, then its operation constitutes a *quantum channel*. It is also known [16, 39] that every quantum channel has at least one stationary mixed state, and we assume Nature selects one.) Finally, given that the incoming CTC bits are now presumed to be in a stationary distribution for the Markov chain M_x, the algorithm $\mathcal{A}$ effectively gets one *sample* from this stationary distribution. Using this sample, the algorithm $\mathcal{A}$ can output its decision on whether or not $x \in L$. When thinking of $\mathcal{A}$ as a BPP-type machine, this decision should be correct with probability at least $\frac{2}{3}$.

1.2 Prior work

Bacon [11] considered the case of a 1-qubit CTC, though his construction actually works equally well with a CTC supporting just 1 classical bit. However, Bacon's model was also more generous in that he allowed 1-bit CTC computations as "subroutines" within polynomial-time algorithms; in effect, he allowed the use of poly(n) many 1-(qu)bit CTCs. Bacon showed that in this model one can efficiently solve any NP problem. Subsequently, Aaronson and Watrous [3, 5] investigated the model in which the CTC supports poly(n) many bits (i.e., $S = 2^{\text{poly}(n)}$ many states). They showed that this model is extremely powerful: if $\mathcal{A}$'s computational power is anywhere between AC^0 and PSPACE (including the most usual choices of BPP or BQP), the result is that the model becomes equivalent in power to PSPACE. Actually, this result was not even the main one in their paper; their main result is that if poly(n) many CTC *qubits* are allowed, then the power of the model is still *only* that of PSPACE.

Regarding the difference between using a 1-bit CTC polynomially many times, and using a poly(n)-bit CTC once, Aaronson [3] remarked, "It is difficult to say which model is the more reasonable!" One can argue that both models are rather impractical in that they require constructing new/wider CTCs as the input length increases.[1] Indeed, the main technical question left open at the end of Aaronson and Watrous's work was to understand the computational power of the more realistic "narrow" CTCs; e.g., one-time-use CTCs that can only transmit a single bit, or a bounded number of bits. In this direction, Say and Yakaryılmaz [31] showed that augmenting standard complexity models with access to a 1-bit CTC is exactly equal in power to augmenting them with "postselection" [4] (defined in Section 3). In particular, this shows that classical randomized computation with a 1-bit CTC is equivalent to the complexity class $\mathsf{BPP_{path}}$, and quantum computation with a 1-bit CTC is equivalent to the complexity class PP. We recall in further detail the class $\mathsf{BPP_{path}}$ in Section 3. For now, suffice it to say that it contains NP and coNP, is likely equal to $\mathsf{P}^{\mathsf{NP}}_{||}$, and is very likely to be much smaller than PSPACE. In particular, randomized computation with access to a 1-bit CTC can efficiently solve the SAT problem; this is discussed below in Example 3.

[1] Bearing in mind the comment concerning practical considerations in the final paragraph of Bacon's work [11].

To summarize, the aforementioned results show that for classical polynomial-time randomized computation, adding a 1-bit CTC gives the power of $\mathsf{BPP_{path}}$ and adding a poly(n)-bit CTC gives the power of PSPACE. What about in between (presuming of course that $\mathsf{BPP_{path}} \neq \mathsf{PSPACE}$)? Sticking with the more "realistic" end of the spectrum, this is the question motivating our work:

▶ **Question.** *Are* 2*-bit (or* 3*-bit,* 4*-bit etc.) CTCs more powerful than* 1*-bit CTCs?*

2 Formal computational complexity statements

In what follows we formally define the complexity model of computing with CTCs. Our definitions are equivalent to those in [5, 31]; however we phrase them differently, in terms of Markov chains. Informally and in brief, $\mathsf{BPP_{CTC}}_{[w]}$ is the class of languages decidable by efficient randomized algorithms that are allowed to set up a 2^w-state Markov chain and then freely get one sample from the chain's stationary distribution.

▶ **Definition 1.** Let M be an S-state Markov chain. A *state-transition oracle* $\mathcal{M}$ for M is any algorithm that takes as input a state $i \in [S]$ and outputs the state resulting from taking one random step in M starting from state i. Most typically we think of $S = 2^w$ and $\mathcal{M}$ as being implemented by a w-bit-input/output *standard randomized circuit*; i.e., one with AND, OR, NOT, and "probability-$\frac{1}{2}$ coin-flip" gates. We might also consider *standard quantum circuits* $\mathcal{M}$ in which Hadamard and Toffoli gates (which are universal [33]) are also used.

▶ **Definition 2.** Let $w = w(n)$ be a "width" parameter. Consider a deterministic polynomial-time Turing Machine $\mathcal{A}$ that, on input $x \in \{0,1\}^n$, outputs the description of two standard randomized circuits, $\mathcal{M}_x$ and $\mathcal{D}_x$. The circuit $\mathcal{M}_x$ should have w input and output bits, thereby defining a state-transition oracle for a Markov chain M_x on $S = 2^w$ states. The "decision circuit" $\mathcal{D}_x$ should have w input bits and one output bit. We suppose computation proceeds as follows: First, an arbitrary stationary distribution π for M_x is chosen. Next, a sample $\boldsymbol{i} \sim \pi$ is chosen from this distribution and is fed as input to $\mathcal{D}_x$. Finally, $\mathcal{D}_x$'s output gate is considered to be the overall output of $\mathcal{A}$'s computation. We define $\mathsf{BPP_{CTC}}_{[w]}$ to be the class of all languages L such that there exists an $\mathcal{A}$ as above with the following property: for every x (and every stationary distribution π for M_x),

$$\Pr_{\boldsymbol{i}}[\mathcal{A} \text{ outputs } 1] \geq \tfrac{2}{3} \quad \text{when } x \in L, \qquad \Pr_{\boldsymbol{i}}[\mathcal{A} \text{ outputs } 1] \leq \tfrac{2}{3} \quad \text{when } x \notin L.$$

We may also analogously define $\mathsf{BQP_{CTC}}_{[w]}$ in case $\mathcal{M}_x$ and $\mathcal{D}_x$ are allowed to be standard quantum circuits.

▶ Remark. We warn the reader that our notation $\mathsf{C_{CTC}}_{[w]}$ is different from that in [5, 31], in that "$\mathsf{CTC}[w]$" signifies a CTC carrying w *classical* bits. We suggest notation such as $\mathsf{BQP_{QCTC}}_{[w]}$ for the case of CTCs carrying w qubits; however we neither define nor consider CTC-qubits in this paper (except in a concluding open problem).

▶ Remark. There is nothing special about considering Markov chains with S states where S is a power of 2. However we stick with the above notation for simplicity, and for consistency with similar complexity class definitions such as that of $\mathsf{P}^{\mathsf{NP}[w]}$ (polynomial-time computation with $2^w - 1$ parallel queries to an NP oracle).

▶ **Example 3.** Following [31], let us show that $\mathsf{NP} \subseteq \mathsf{BPP_{CTC}}_{[1]}$. Equivalently, we illustrate how SAT can be solved by a "1-bit CTC algorithm" $\mathcal{A}$, which can set up a 2-state Markov chain and get a sample from its stationary distribution. On input an n-variable CNF formula ϕ, algorithm $\mathcal{A}$ constructs a state-transition circuit $\mathcal{M}_\phi$ for a certain 2-state Markov chain M_ϕ. Think of state 0 of M_ϕ as meaning "no evidence that ϕ is satisfiable" and state 1 as meaning "evidence that ϕ is satisfiable". The operation of $\mathcal{M}_\phi$ is as follows: On input state i, $\mathcal{M}_\phi$ first chooses a uniformly random string $\boldsymbol{y} \in \{0,1\}^n$ and checks if it satisfies ϕ. If $\boldsymbol{y}$ is satisfying, $\mathcal{M}_\phi$ outputs state 1. If $\boldsymbol{y}$ is unsatisfying, then $\mathcal{M}_\phi$ outputs state 0 with probability $\epsilon := 2^{-n^2}$ and outputs its input state i with probability $1-\epsilon$. It is clear that $\mathcal{A}$ can write down $\mathcal{M}_\phi$'s description in deterministic polynomial time. One can now check that the resulting Markov chains M_ϕ are as follows:

$$\text{if } \phi \text{ is satisfiable, } M_\phi = \begin{bmatrix} 1-2^{-n} & 2^{-n} \\ \epsilon' & 1-\epsilon' \end{bmatrix} \qquad (\text{where } \epsilon' := (1-2^{-n})\epsilon \approx 2^{-n^2});$$

$$\text{if } \phi \text{ is unsatisfiable, } M_\phi = \begin{bmatrix} 1 & 0 \\ \epsilon & 1-\epsilon \end{bmatrix}.$$

Now if ϕ is unsatisfiable, state 0 is absorbing and it's clear that the (unique) stationary distribution π of M_ϕ is entirely concentrated on state 0. On the other hand, suppose ϕ is satisfiable. Then since the $0 \to 1$ transition probability of M_ϕ is much higher (relatively speaking) than the $1 \to 0$ transition probability, the long-term (i.e., stationary) distribution π of M_ϕ will be almost entirely concentrated on state 1. (More precisely, π will put only probability $\frac{\epsilon'}{2^{-n}+\epsilon'} \approx 2^{-n^2+n}$ on state 0.) We now stipulate that for every ϕ, algorithm $\mathcal{A}$ outputs the same 1-bit decision circuit $\mathcal{D}_\phi$, which on input $\boldsymbol{i} \sim \pi$ simply outputs $\boldsymbol{i}$. From the above discussion, we see that this correctly indicates whether $\phi \in \mathsf{SAT}$ except with negligible error probability.

The following theorems concerning $\mathsf{BPP}_{\mathsf{CTC}[w]}$ have previously been shown:

▶ **Theorem 4.** *(Aaronson–Watrous [5].)* $\mathsf{BPP}_{\mathsf{CTC}[\mathrm{poly}(n)]} = \mathsf{PSPACE}$.
(Indeed $\mathsf{P}_{\mathsf{CTC}[\mathrm{poly}(n)]} = \mathsf{BQP}_{\mathsf{QCTC}[\mathrm{poly}(n)]} = \mathsf{PSPACE}$.*)*

▶ **Theorem 5.** *(Say–Yakaryılmaz [31].)* $\mathsf{BPP}_{\mathsf{CTC}[1]} = \mathsf{BPP}_{\mathsf{path}}$.
(Indeed, adding 1 *CTC-bit generally confers the power of "postselection", discussed in Section 3. For example, it also holds that* $\mathsf{BQP}_{\mathsf{CTC}[1]} = \mathsf{PostBQP} = \mathsf{PP}$.*)*

As mentioned in Section 1.1 (and discussed further in Section 3), $\mathsf{BPP}_{\mathsf{path}}$ is likely equal to $\mathsf{P}^{\mathsf{NP}}_{||} = \mathsf{P}^{\mathsf{NP}[O(\log n)]}$, and is very likely to be much smaller than PSPACE.

2.1 Our theorem

Paraphrasing the above two theorems, we have that Markov chains with 2 states ($w = 1$) give the power of $\mathsf{BPP}_{\mathsf{path}}$, and Markov chains with exponentially many states ($w = \mathrm{poly}(n)$) give the power of PSPACE. What about in between? Are 3-state or 4-state ($w = 2$) Markov chains more powerful than 2-state chains? To take an analogy from another family of complexity classes, we remind the reader that it's widely believed that $\mathsf{P}^{\mathsf{NP}[1]} \subsetneq \mathsf{P}^{\mathsf{NP}[2]} \subsetneq \mathsf{P}^{\mathsf{NP}[3]} \subsetneq \cdots$ The main result of this paper is that in apparent contrast, "the hierarchy collapses" for $\mathsf{BPP}_{\mathsf{CTC}[w]}$; polynomially many states ($w = O(\log n)$) confer no more advantage than 2 states.

▶ **Main theorem.** $\mathsf{BPP}_{\mathsf{CTC}[O(\log n)]} \subseteq \mathsf{PostBPP} = \mathsf{BPP}_{\mathsf{path}}$*; thus*

$$\mathsf{BPP}_{\mathsf{CTC}[1]} = \mathsf{BPP}_{\mathsf{CTC}[2]} = \mathsf{BPP}_{\mathsf{CTC}[3]} = \cdots = \mathsf{BPP}_{\mathsf{CTC}[O(\log n)]} = \mathsf{BPP}_{\mathsf{path}}.$$

It will be clear from our proof that more generally, $O(\log n)$ CTC bits still only confer the power of postselection, and in particular $\mathsf{BQP}_{\mathsf{CTC}[O(\log n)]} = \mathsf{PostBQP} = \mathsf{PP}$. Our main theorem may also be seen as further demonstration of the robustness and naturalness of the class $\mathsf{BPP}_{\mathsf{path}}$.

2.2 Proof techniques

Here we briefly outline the proof of our theorem, with the actual proof being given in Section 4. Let's return to Example 3, which shows that SAT $\in \mathsf{BPP}_{\mathsf{CTC}[1]}$. One might ask, why doesn't the proof show that SAT $\in$ BPP? After all, the algorithm $\mathcal{A}$ simply constructs a 2-state Markov chain M and then takes a sample from its stationary distribution. Why doesn't $\mathcal{A}$ simply exactly solve for M's stationary distribution? The trouble of course is that even though $\mathcal{A}$ constructed M itself, in some sense M is still only "implicitly defined" from $\mathcal{A}$'s point of view. $\mathcal{A}$ cannot directly access the transition probabilities of M (doing so requires $\mathcal{A}$ to solve an NP-complete problem); rather, $\mathcal{A}$ can only "simulate" M, by use of the state-transition matrix $\mathcal{M}$ it constructed. Naively, this still might not seem like a problem; given the ability to simulate M, couldn't $\mathcal{A}$ find a (near-)stationary distribution π for M simply by simulating it for a long time? The trouble here is that even though M only has 2 states, it has some transition probabilities that are "exponentially small" (in n). Furthermore, the stationary distribution of M can be extremely sensitive to the *relative* exponential smallness of these transition probabilities – Example 3 illustrates exactly this.

Our proof that $\mathsf{BPP}_{\mathsf{CTC}[w]} \subseteq \mathsf{PostBPP} = \mathsf{BPP}_{\mathsf{path}}$ for $w \leq O(\log n)$ in some sense follows Say and Yakaryılmaz's proof [31] in the case of $w = 1$. They essentially observed that using the power of postselection (discussed further in Section 3), a randomized algorithm can get an exact sample from the stationary distribution of a Markov chain given only a state-transition oracle for it. Their proof of this was greatly facilitated by the fact that 2-state Markov chains are easy to analyze: If the $0 \to 1$ transition probability is p and the $1 \to 0$ transition probability is q, then the stationary distribution is $\pi = \begin{bmatrix} \frac{q}{p+q} \\ \frac{p}{p+q} \end{bmatrix}$. (This presumes we don't have $p = q = 0$, an important issue that we discuss later.) For our main theorem, we need a similar postselecting algorithm for general poly(n)-state Markov chains. The key technical tool for this will be the *Markov Chain Tree Theorem*, apparently first proved by Hill [23], and called by Aldous [6] "the most often rediscovered result in probability theory"; see also [36, 34, 24, 25, 7, 14, 28]. We state here the version for irreducible chains:

▶ **Markov Chain Tree Theorem.** *Let M be an S-state irreducible Markov chain with transition matrix $(p_{ij})_{i,j \in [S]}$. Let G_M be the underlying strongly connected digraph for M in which (i, j) is a directed edge if and only if $p_{ij} > 0$. Recall that a* rooted arborescence *T in G_M is a collection of edges forming a rooted spanning tree in which all edges are directed toward the root vertex. We write $\|T\| = \prod_{(i,j) \in T} p_{ij}$. Let $\mathcal{T}_i$ denote the set of all arborescences in G_M rooted at $i \in [S]$, and write $\mathcal{T} = \cup_i \mathcal{T}_i$. Then if π denotes the (unique) stationary distribution of M, we have the formula $\pi_i = \left(\sum_{T \in \mathcal{T}_i} \|T\|\right) \Big/ \left(\sum_{T \in \mathcal{T}} \|T\|\right)$.*

We add that this theorem plays an important role in the theory of exact sampling from unknown Markov chains [8, 27, 29, 38]. That theory is concerned with a problem similar to ours; however, there are two main differences: i) That theory involves only traditional algorithms, and therefore by necessity the running time may be exponential if the chain's mixing time is exponential. By contrast, we are using postselecting algorithms and therefore have the chance to run in polynomial time. ii) That theory is concerned with *exact* sampling from the stationary distribution. By contrast, we actually only need approximate sampling from the stationary distribution.

Finally, we mention one challenge for our proof that at first seems like a technicality but in fact proves to be quite a nuisance: There is no promise in the definition of $\mathsf{BPP}_{\mathsf{CTC}[w]}$ that the Markov chains M_x be irreducible. This is precisely the "$p = q = 0$ issue" elided in the discussion of 2-state stationary distributions above. We overcome this difficulty by proving a somewhat technical lemma that allows us to perturb general Markov chains into irreducible ones.

2.3 Outline of the remainder of the paper

The aforementioned technical lemma on Markov chain perturbations, which allows us to work only with irreducible Markov chains, is omitted for reasons of space; it can be found in the arXiv version of the paper. In Section 3 we recall $\mathsf{BPP}_{\mathsf{path}}$ and postselection in more detail, and we also describe the "restarting" view of postselection (from [40]) that will be helpful in the proof of our main theorem. Finally, we give the proof of the main theorem in Section 4, and then end with an open question.

3 $\mathsf{BPP}_{\mathsf{path}}$, postselection, and restarts

In this section we describe three different viewpoints on the class $\mathsf{BPP}_{\mathsf{path}}$.

The complexity class $\mathsf{BPP}_{\mathsf{path}}$ was originally defined by Han, Hemaspaandra, and Thierauf [22, 21], in a paper also concerned with certain cryptographic problems. (It was also independently defined much later in a paper by Aspnes, Fischer, Fischer, Kao, and Kumar [9, 10] on the computational complexity of the stock market.) We quote Fortnow's explanation of the original definition when he named it "Complexity Class of the Week" [17]:

"Let us call a nondeterministic Turing machine M balanced if for every input x, all of its computational paths have the same length. [We can define the] class BPP as follows: L is in BPP if there is a balanced nondeterministic polynomial-time M such that:

- If x is in L then there are at least twice as many accepting as rejecting paths of $M(x)$.
- If x is not in L then there are at least twice as many rejecting as accepting paths of $M(x)$.

Suppose we use the same definition without the "balanced" requirement. This gives us the class $\mathsf{BPP}_{\mathsf{path}}$."

Interestingly, the analogous class "$\mathsf{PP}_{\mathsf{path}}$" – for which $x \in L$ iff $M(x)$ has more accepting than rejecting (unbalanced) paths – was defined much earlier in Simon's 1975 thesis [35]. Simon showed that $\mathsf{PP}_{\mathsf{path}}$ is equal to the class PP (which had recently been defined by Gill [19]). By way of contrast, $\mathsf{BPP}_{\mathsf{path}}$ is very unlikely to equal BPP, as it is known [22] that $\mathsf{BPP}_{\mathsf{path}}$ contains both MA and $\mathsf{P}^{\mathsf{NP}}_{||}$. $\mathsf{BPP}_{\mathsf{path}}$ is also known [22] to be contained in $\mathsf{BPP}^{\mathsf{NP}}_{||}$. Indeed, under the standard complexity assumptions used to derandomize AM, Shaltiel and Umans [32] showed that $\mathsf{BPP}_{\mathsf{path}} = \mathsf{P}^{\mathsf{NP}}_{||}$. For a related class known as SBP, which sits between MA and $\mathsf{BPP}_{\mathsf{path}}$, see [13, 12].

Another characterization of $\mathsf{BPP}_{\mathsf{path}}$ was given by Aaronson, via the notion of adding *postselection* (see [4]) to a complexity class. Suppose that you have a probabilistic algorithm that can end in three kinds of final states: accepting, rejecting, and indecisive. We assume the probability of ending in a decisive state is guaranteed to be nonzero. "Postselection" refers to the (nonrealistic) ability to *condition* the computation on ending in a decisive state. This yields probabilistic computation with just the usual two kinds of final states. For example, one says that $L \in \mathsf{PostBPP}$ if there is a polynomial-time randomized Turing machine as described above which, for each input x, gives the correct answer about $x \in L$ with probability at least $\frac{2}{3}$, *conditioned* on not ending in an indecisive state. More generally, if C is a probabilistic or quantum complexity class, PostC is the class of languages decided by C-machines with the ability to postselect on ending in a decisive state. In [1], Aaronson proved that $\mathsf{PostBQP} = \mathsf{PP}$; later [2], he observed that $\mathsf{PostBPP} = \mathsf{BPP_{path}}$.

In the derivation of our main theorem we will prefer a third perspective on $\mathsf{BPP_{path}}$ and postselection, introduced by Yakaryılmaz and Say [40]: that of randomized algorithms with *restarts*. For some probabilistic complexity class C, suppose again that we have C-machines that can end in one of three states: accept, reject, or indecisive. We think of the third state as the *restart* state, imagining that whenever the C-machine enters such a state, it immediately restarts its computation from the initial configuration, using no information that it may have gathered up to that point.[2] As observed in [40], the class of languages decided by such a machine is again PostC. In particular, $\mathsf{BPP_{path}}$ is the class of languages that are decided by bounded-error probabilistic polynomial-time Turing machines with this ability to restart. This perspective seems most useful for algorithm design. As an illustration, we believe it is fairly "obvious" that the following restarting-algorithm decides SAT with very high probability, thereby showing $\mathsf{NP} \subseteq \mathsf{BPP_{path}}$:

"On input formula ϕ with n variables, randomly choose an assignment $\boldsymbol{y}$.
If $\boldsymbol{y}$ satisfies ϕ, accept. (1)
Otherwise, restart with probability $1 - 2^{-n^2}$ and reject with probability 2^{-n^2}."

The reader may compare (1) with the 1-bit CTC algorithm for SAT from Example 3, which also shows $\mathsf{NP} \subseteq \mathsf{BPP_{path}}$ in light of Say and Yakaryılmaz's Theorem 5, $\mathsf{BPP_{CTC[1]}} = \mathsf{BPP_{path}}$.

▶ Remark. For all three definitions of $\mathsf{BPP_{path}}$ described above, it is easy to see that the "$\frac{2}{3}$ cutoff" for success could equivalently be an "α cutoff" for any fixed constant $\frac{1}{2} < \alpha < 1$, just as is the case for the class BPP.

3.1 Remarks on random coins for $\mathsf{BPP_{path}}$ algorithms

In informal descriptions of randomized algorithms, it's typical to make statements like, "Next, with probability $\frac{1}{3}$ the algorithm…" Such statements sweep a well-known, minor detail under the rug; namely, the traditional BPP model (based on nondeterministic branching) only has "access" to probability-$\frac{1}{2}$ coin flips. Of course, this is not an essential problem, since one can simulate a $\frac{1}{3}$-biased coin flip to error δ in time $O(\log \frac{1}{\delta})$, and δ needn't be smaller than $1/\text{poly}(n)$. Here we remark that in the context of *restarting* algorithms, the problem is not just inessential, it's literally no problem at all:

▶ **Lemma 6.** *A restarting randomized algorithm can simulate a p-biased coin flip* exactly *in time $O(\langle p \rangle)$, and can simulate a uniformly random draw from $[n]$* exactly *in time $O(\log n)$.*

Proof. We give the simplest example, leaving the general case for the reader. Suppose we wish to draw $\boldsymbol{r} \sim [3]$ uniformly at random. We toss two probability-$\frac{1}{2}$ coins, forming a 2-bit integer $0 \leq \boldsymbol{r} \leq 3$. Then if $\boldsymbol{r} = 0$, we restart. ◀

On the other hand, it's also important to remember that time $\Omega(\langle p \rangle)$ is also *required* to flip a p-biased coin; for example, in algorithm (1) the step "reject with probability 2^{-n^2} else restart" takes n^2 time steps.

[2] Note that the new algorithm obtained in this manner will in general have unbounded runtime, even if C is a time-bounded class.

4 Proof of the main theorem

We now give the proof of our main theorem, that $\mathsf{BPP}_{\mathsf{CTC}[O(\log n)]} \subseteq \mathsf{BPP}_{\mathsf{path}}$.

Proof. Let $L \in \mathsf{BPP}_{\mathsf{CTC}[O(\log n)]}$; say L is defined by algorithm $\mathcal{A}$ as in Definition 2. Thus there are constants $c_1, c_2, c_3 \in \mathbb{N}$ such that on inputs $x \in \{0,1\}^n$, algorithm $\mathcal{A}$ outputs state-transition circuits $\mathcal{M}_x$ of size $O(n^{c_1})$ defining Markov chains M_x on $S = O(n^{c_2})$ states, as well as decision circuits $\mathcal{D}_x$ of size $O(n^{c_3})$. Further, we have that for each x, if π is *any* stationary distribution for M_x and $\boldsymbol{i} \sim \pi$, then

$$\mathbf{Pr}[\mathcal{D}_x(\boldsymbol{i}) = 1_{\{x \in L\}}] \geq \tfrac{2}{3}. \tag{2}$$

Our goal will be to define a polynomial-time randomized restarting algorithm $\mathcal{R}$ that has

$$\mathbf{Pr}[\mathcal{R}(x) = 1_{\{x \in L\}}] \geq 0.65 \tag{3}$$

for all x. As discussed in Section 3, this will show that $L \in \mathsf{BPP}_{\mathsf{path}}$, as required.

On input $x \in \{0,1\}^n$, the first step of algorithm $\mathcal{R}$ involves invoking a technical lemma to convert to an irreducible Markov chain. This lemma (whose proof is omitted from this extended abstract) involves replacing the transition matrix K of the chain with $K' = (1-\epsilon)K + \epsilon\frac{1}{S}J$, where S is the number of states in the chain and J is the all-1's matrix. Here the exact value $\epsilon = 2^{-\mathrm{poly}(n)}$ will be described later. More precisely, $\mathcal{R}$ first simulates $\mathcal{A}$ to get state-transition oracle circuit $\mathcal{M}_x$ for Markov chain M_x. It then constructs a state-transition oracle circuit $\mathcal{M}'_x$ for the irreducible perturbed chain M'_x, using the description of $\mathcal{M}_x$ in a black-box fashion. Let π' denote the stationary distribution for M'_x and let K denote the transition matrix for M_x. By definition, K is square matrix of dimension $S \leq O(n^{c_2})$ in which each entry is an integer multiple of $2^{-\mathrm{size}(\mathcal{M}_x)} = 2^{-O(n^{c_1})}$. It follows that $\langle K \rangle \leq O(n^{c_4})$ for some constant $c_4 \in \mathbb{N}$. We will now specify that $\epsilon = 2^{-bn^b}$ for a sufficiently large constant b depending on c_1, c_2, and a constant from the aforementioned technical lemma. Then that lemma implies

$$\|\pi - \pi'\|_1 \leq .01 \tag{4}$$

for *some* stationary distribution π of M_x. It is easy to see that with this choice of ϵ, algorithm $\mathcal{R}$ can construct $\mathcal{M}'_x$ from $\mathcal{M}_x$ in $\mathrm{poly}(n)$ time. (Here we use the fact that ϵ is "only" exponentially small; cf. the last paragraph in Section 3.1.)

The remainder of the proof is devoted to showing that $\mathcal{R}$ can obtain an *exact* sample $\boldsymbol{r} \sim \pi'$. Having shown this, we only need to let $\mathcal{R}$ simulate $\mathcal{A}$ to get $\mathcal{D}_x$, and then output $\mathcal{D}_x(\boldsymbol{r})$. Then combining (4) with (2) shows that (3) holds for all x, as required.

We now exhibit the subroutine which the restarting-algorithm $\mathcal{R}$ will use to obtain an exact sample $\boldsymbol{r} \sim \pi'$ (the unique stationary distribution of irreducible chain M'_x):

- Choose a uniformly random labeled, rooted, undirected tree $\boldsymbol{T}$ on vertex set $[S]$. This can be done exactly (i.e., with each $\boldsymbol{T}$ occurring with probability $1/S^{S-1}$) in $\mathrm{poly}(S) = \mathrm{poly}(n)$ time, by choosing a uniformly random Prüfer Code [30, 37] in $[S]^{S-2}$, converting it to a tree $\boldsymbol{T}$, and then choosing a random vertex $\boldsymbol{r}$ of $\boldsymbol{T}$ to be the root.[3]
- Make $\boldsymbol{T}$ into a rooted arborescence $\vec{\boldsymbol{T}}$ by directing all edges toward the root $\boldsymbol{r}$.
- For each directed edge $(i,j) \in \vec{\boldsymbol{T}}$, simulate $\mathcal{M}'_x(i)$ and "check" if the output is j. If the check fails, restart.
- If all $S-1$ checks pass, halt with output $\boldsymbol{r}$.

[3] Here we may use restarting to get exactly random samples from $[S]$; see Lemma 6.

The fact that this subroutine restarts with probability strictly less than 1 follows from the fact that M'_x is irreducible; indeed, its underlying digraph $G = G_{M'_x}$ is the complete digraph. The probability P_r that this subroutine outputs $\boldsymbol{r} = r$ without encountering any restarts is precisely

$$P_r = \sum_{r\text{-rooted arborescences } T} \frac{1}{S^{S-1}} \prod_{(i,j)\in T} p_{ij},$$

where p_{ij} denotes the transition probability from i to j in the Markov chain M'_x. It follows that the probability of $\mathcal{R}$ finally outputting r (when restarts are taken into account) is

$$\frac{P_r}{\sum_{r\in[S]} P_r} = \frac{\sum_{r\text{-rooted arborescences } T} \prod_{(i,j)\in T} p_{ij}}{\sum_{\text{arborescences } T} \prod_{(i,j)\in T} p_{ij}}.$$

By the Markov Chain Tree Theorem, this is indeed precisely the probability $\pi'(r)$ of r under the stationary distribution of M'_x. ◀

We conclude this section by observing that besides the power of restarting, algorithm $\mathcal{R}$ only really needed the power to simulate the state-transition circuits $\mathcal{M}_x$ and the decision circuits $\mathcal{D}_x$. For example, if these were standard quantum circuits, it would suffice for $\mathcal{R}$ to be a quantum algorithm. Thus we may also conclude $\mathsf{BQP}_{\mathsf{CTC}[O(\log n)]} \subseteq \mathsf{PostBQP} = \mathsf{PP}$.

We also add that Say and Yakaryılmaz [31] studied various models of *finite automata* augmented with 1-bit CTCs; they showed that this augmentation causes both probabilistic and quantum finite automata to become as powerful as their respective postselected versions. The technique used in the proof of our main result can be simplified easily to show that no additional gain arises when these machines are augmented with larger constant-width CTCs.

5 Conclusion

A very interesting open question left by our work is one also raised at the end of [31]:

What is the computational power conferred by 1 time-traveling *qubit*?

Answering this question precisely would seem to require a good understanding of stationary density matrices for 1-qubit quantum channels. As mentioned in Section 1.1, we are not aware of any work showing that time-traveling qubits confer more computational power than time-traveling bits.

Acknowledgment. The first author would like to thank the Boğaziçi University Computer Engineering Department for their hospitality.

References

1 Scott Aaronson. Is quantum mechanics an island in Theoryspace? Technical Report quant-ph/0401062, arXiv, 2004.

2 Scott Aaronson. *Limits on Efficient Computation in the Physical World.* PhD thesis, University of California, Berkeley, 2004.

3 Scott Aaronson. NP-complete problems and physical reality. *ACM SIGACT News*, 36(1):30–52, 2005.

4 Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A*, 461(2063):3473–3482, 2005.

5 Scott Aaronson and John Watrous. Closed timelike curves make quantum and classical computing equivalent. *Proceedings of the Royal Society A*, 465(2102):631–647, 2009.

6 David Aldous. Reversible Markov chains and random walks on graphs, 2002. `http://www.stat.berkeley.edu/~aldous/RWG/book.pdf`.

7 Venkat Anantharam and Pantelis Tsoucas. A proof of the Markov chain tree theorem. *Statistics & Probability Letters*, 8(2):189–192, 1989.

8 Søren Asmussen, Peter Glynn, and Hermann Thorisson. Stationarity detection in the initial transient problem. *ACM Transactions on Modeling and Computer Simulation*, 2(2):130–157, 1992.

9 James Aspnes, David Fischer, Michael Fischer, Ming-Yang Kao, and Alok Kumar. Towards understanding the predictability of stock markets from the perspective of computational complexity. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 745–754, 2001.

10 James Aspnes, David Fischer, Michael Fischer, Ming-Yang Kao, and Alok Kumar. Towards understanding the predictability of stock markets from the perspective of computational complexity. In *New Directions in Statistical Physics*, pages 129–151. Springer Berlin Heidelberg, 2004.

11 Dave Bacon. Quantum computational complexity in the presence of closed timelike curves. *Physical Review A*, 70(3):032309, 2004.

12 Elmar Böhler, Christian Glaßer, and Daniel Meister. Small bounded-error computations and completeness. Technical Report 69, Electronic Colloquium on Computational Complexity, 2003.

13 Elmar Böhler, Christian Glaßer, and Daniel Meister. Error-bounded probabilistic computations between MA and AM. *Journal of Computer and System Sciences*, 72(6):1043–1076, 2006.

14 Andrei Broder. Generating random spanning trees. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 442–447, 1989.

15 Todd Brun. Computers with closed timelike curves can solve hard problems efficiently. *Foundations of Physics Letters*, 16(3):245–253, 2003.

16 David Deutsch. Quantum mechanics near closed timelike lines. *Physical Review D*, 44(10):3197–3217, 1991.

17 Lance Fortnow. Complexity class of the week: $\mathrm{BPP}_{\mathrm{path}}$, February 2003. `http://blog.computationalcomplexity.org/2003/02/complexity-class-of-week-bpppath.html`.

18 John Friedman, Michael Morris, Igor Novikov, Fernando Echeverria, Gunnar Klinkhammer, Kip Thorne, and Ulvi Yurtsever. Cauchy problem in spacetimes with closed timelike curves. *Physical Review D*, 42(6):1915–1930, 1990.

19 John Gill. Computational complexity of probabilistic Turing machines. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, pages 91–95, 1974.

20 Kurt Gödel. An example of a new type of cosmological solutions of Einstein's field equations of gravitation. *Reviews of Modern Physics*, 21(3):447–450, 1949.

21 Yenjo Han, Lane Hemaspaandra, and Thomas Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.

22 Yenjo Hem, Lane Hemaspaandra, and Thomas Thierauf. Threshold computation and cryptographic security. In *Proceedings of the 4th Annual International Symposium on Algorithms and Computation*, pages 230–239, 1993.

23 Terrell Hill. Studies in irreversible thermodynamics IV: diagrammatic representation of steady state fluxes for unimolecular systems. *Journal of Theoretical Biology*, 10(3):442–459, 1966.

24 Hans-Helmut Kohler and Eva Vollmerhaus. The frequency of cyclic processes in biological multistate systems. *Journal of Mathematical Biology*, 9(3):275–290, 1980.

25 F. Thomson Leighton and Ronald Rivest. The Markov Chain Tree Theorem. Technical Report TM-249, Massachusetts Institute of Technology, 1983.

26 Seth Lloyd, Lorenzo Maccone, Raul Garcia-Patron, Vittorio Giovannetti, and Yutaka Shikano. The quantum mechanics of time travel through post-selected teleportation. Technical Report 1007.2615, arXiv, 2010.

27 László Lovász and Peter Winkler. Exact mixing in an unknown Markov chain. *Electronic Journal of Combinatorics*, 2:Research Paper 15, 1995.

28 Piotr Pokarowski. Directed forests with application to algorithms related to Markov chains. *Applicationes Mathematicae*, 26(4):395–414, 1999.

29 James Propp and David Wilson. How to get a perfectly random sample from a generic markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, 27(2):170–217, 1998.

30 Heinz Prüfer. Neuer Beweis eines Satzes über Permutationen. *Archiv der Mathematik und Physik*, 27:742–744, 1918.

31 A. C. Cem Say and Abuzer Yakaryılmaz. Computation with multiple CTCs of fixed length and width. *Natural Computing*, 11(4):579–594, 2012.

32 Ronen Shaltiel and Christopher Umans. Pseudorandomness for approximate counting and sampling. *Computational Complexity*, 15(4):298–341, 2006.

33 Yaoyun Shi. Both Toffoli and controlled-NOT need little help to do universal quantum computing. *Quantum Information & Computation*, 3(1):84–92, 2003.

34 Bruno Shubert. A flow-graph formula for the stationary distribution of a Markov chain. *Institute of Electrical and Electronics Engineers. Transactions on Systems, Man, and Cybernetics*, SMC-5(5):565–566, 1975.

35 Janos Simon. *On some central problems in computational complexity.* PhD thesis, Cornell University, 1975. TR 75-224.

36 Alexander Wentzell and Mark Freidlin. On small random perturbations of dynamical systems. *Russian Mathematical Surveys*, 25(1):1–55, 1970.

37 Wikipedia. Prüfer sequence, July 2014. `http://en.wikipedia.org/wiki/Prufer_sequence`.

38 David Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 296–303, 1996.

39 Michael Wolf. Quantum channels & operations: guided tour, 2012. `http://www-m5.ma.tum.de/foswiki/pub/M5/Allgemeines/MichaelWolf/QChannelLecture.pdf`.

40 Abuzer Yakaryılmaz and A. C. Cem Say. Proving the power of postselection. *Fundamenta Informaticae*, 123(1):107–134, 2013.

New Bounds for the Garden-Hose Model*

Hartmut Klauck[1] and Supartha Podder[2]

1 **CQT and Nanyang Technological University, Singapore**
hklauck@gmail.com

2 **CQT Singapore**
supartha@gmail.com

Abstract

We show new results about the garden-hose model. Our main results include improved lower bounds based on non-deterministic communication complexity (leading to the previously unknown $\Theta(n)$ bounds for Inner Product mod 2 and Disjointness), as well as an $O(n \cdot \log^3 n)$ upper bound for the Distributed Majority function (previously conjectured to have quadratic complexity). We show an efficient simulation of formulae made of AND, OR, XOR gates in the garden-hose model, which implies that lower bounds on the garden-hose complexity $GH(f)$ of the order $\Omega(n^{2+\epsilon})$ will be hard to obtain for explicit functions. Furthermore we study a time-bounded variant of the model, in which even modest savings in time can lead to exponential lower bounds on the size of garden-hose protocols.

1998 ACM Subject Classification F.1.1 Models of Computation, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Space Complexity, Communication Complexity, Garden-Hose Model

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.481

1 Introduction

1.1 Background: The Model

Recently, Buhrman et al. [4] proposed a new measure of complexity for finite Boolean functions, called *garden-hose complexity*. This measure can be viewed as a type of distributed space complexity, and while its motivation is mainly in applications to position based quantum cryptography, the playful definition of the model is quite appealing in itself. Garden-hose complexity can be viewed as a natural measure of space, when two players with private inputs compute a Boolean function cooperatively. Space-bounded communication complexity has been investigated before [2, 7, 9] (usually for problems with many outputs), and recently Brody et al. [3] have studied a related model of space bounded communication complexity for Boolean functions (see also [17]). In this context the garden-hose model can be viewed as a memoryless model of communication that is also reversible.

To describe the garden-hose model let us consider two neighbors, Alice and Bob. They own adjacent gardens which happen to have s empty water pipes crossing their common boundary. These pipes are the only means of communication available to the two. Their goal is to compute a Boolean function on a pair of private inputs, using water and the pipes across their gardens as a means of communication.[1]

* This work is funded by the Singapore Ministry of Education (partly through the Academic Research Fund Tier 3 MOE2012-T3-1-009) and by the Singapore National Research Foundation.

[1] It should be mentioned that even though Alice and Bob choose to not communicate in any other way, their intentions are not hostile and neither will deviate from a previously agreed upon protocol.

A garden-hose protocol works as follows: There are s shared pipes. Alice takes some pieces of hose and connects pairs of the open ends of the s pipes. She may keep some of the ends open. Bob acts in the same way for his end of the pipes. The connections Alice and Bob place depend on their local inputs x, y, and we stress that every end of a pipe is only connected to at most one other end of a pipe (meaning no Y-shaped pieces of hose may be used to split or combine the flow of water). Finally, Alice connects a water tap to one of those open ends on her side and starts the water. Based on the connections of Alice and Bob, water flows back and forth through the pipes and finally ends up spilling on one side.

If the water spills on Alice's side we define the output to be 0. Otherwise, the water spills on Bob's side and the output value is 1. It is easy to see that due to the way the connections are made the water must eventually spill on one of the two sides, since cycles are not possible.

Note that the pipes can be viewed as a communication channel that can transmit $\log s$ bits, and that the garden-hose protocol is memoryless, i.e., regardless of the previous history, water from pipe i always flows to pipe j if those two pipes are connected. Furthermore computation is reversible, i.e., one can follow the path taken by the water backwards (e.g. by sucking the water back).

Buhrman et al. [4] have shown that it is possible to compute every function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ by playing a garden-hose game. A garden-hose protocol consists of the scheme by which Alice chooses her connections depending on her private input $x \in \{0,1\}^n$ and how Bob chooses his connections depending on his private input $y \in \{0,1\}^n$. Alice also chooses the pipe that is connected to the tap. The protocol computes a function f, if for all inputs with $f(x,y) = 0$ the water spills on Alice's side, and for all inputs with $f(x,y) = 1$ the water spills on Bob's side.

The size of a garden-hose protocol is the number s of pipes used. The garden-hose complexity $GH(f)$ of a function $f(x,y)$ is the minimum number of pipes needed in any garden-hose game that computes the value of f for all x and y such that $f(x,y)$ is defined.

The garden-hose model is originally motivated by an application to quantum position-verification schemes [4]. In this setting the position of a prover is verified via communications between the prover and several verifiers. An attack on such a scheme is performed by several provers, none of which are in the claimed position. [4] proposes a protocol for position-verification that depends on a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, and a certain attack on this scheme requires the attackers to share as many entangled qubits as the garden-hose complexity of f. Hence all f with low garden-hose complexity are not suitable for this task, and it becomes desirable to find explicit functions with large garden-hose complexity.

Buhrman et al. [4] prove a number of results about the garden-hose model:

- Deterministic one-way communication complexity can be used to show lower bounds of up to $\Omega(n/\log n)$ for many functions.
- For the Equality problem they refer to a bound of $GH(\mathit{Equality}) = \Theta(n)$ shown by Pietrzak (the proof implicitly uses the fooling set technique from communication complexity [10] [personal communication]).
- They argue that super-polynomial lower bounds for the garden-hose complexity of a function f imply that the function cannot be computed in Logspace, making such bounds hard to prove for 'explicit' functions.
- They define randomized and quantum variants of the model and show that randomness can be removed at the expense of multiplying size by a factor of $O(n)$ (for quantum larger gaps are known).
- Via a counting argument it is easy to see that most Boolean functions need size $GH(f) = 2^{\Omega(n)}$.

Very recently Chiu et al. [5] have improved the upper bound for the Equality function to $1.359n$ from the previously known $2n$ bound [4].

1.2 Our Results

We study garden-hose complexity and establish several new connections with well studied models like communication complexity, permutation branching programs, and formula size.

We start by showing that non-deterministic communication complexity gives lower bounds on the garden-hose complexity of any function f. This improves the lower bounds of $\Omega(\frac{n}{\log n})$ for several important functions like Inner Product, Disjointness to $\Omega(n)$.

We observe that any 2-way deterministic communication protocol can be converted to a garden-hose protocol so that the complexity $GH(f)$ is upper bounded by the *size* of the protocol tree of the communication protocol.

We then turn to comparing the model to another nonuniform notion of space complexity, namely branching programs. We show how to convert any *permutation branching program* to a garden-hose protocol with only a constant factor loss in size.

The most important application of this simulation is that it allows us to find a garden-hose protocol for the distributed Majority function, $DMAJ(x, y) = 1$ iff $\sum_{i=1}^{n}(x_i \cdot y_i) \geq \frac{n}{2}$, that has size $O(n \cdot \log^3 n)$, disproving the conjecture in [4] that this function has complexity $\Omega(n^2)$.

Using the garden-hose protocols for Majority, Parity, AND, OR, we show upper bounds on the composition of functions with these.

We then show how to convert any Boolean formula with AND, OR, XOR gates to a garden-hose protocol with a small loss in size. In particular, any formula consisting of arbitrary fan-in 2 gates only can be simulated by a garden-hose protocol with a constant factor loss in size. This result strengthens the previous observation that explicit super-polynomial lower bounds for $GH(f)$ will be hard to show: even bounds of $\Omega(n^{2+\epsilon})$ would improve on the long-standing best lower bounds on formula size due to Nečiporuk from 1966 [12]. We can also simulate formulae including a limited number of Majority gates of arbitrary fan-in, so one might be worried that even super-linear lower bounds could be difficult to prove. We argue, however, that for formulae using arbitrary symmetric gates we can still get near-quadratic lower bounds using a Nečiporuk-type method. Nevertheless we have to leave super-linear lower bounds on the garden-hose complexity as an open problem.

Next we define a notion of *time* in garden-hose protocols and prove that for any function f, if we restrict the number of times water can flow through pipes to some value k, we have $GH_k(f) = \Omega(2^{D_k(f)/k})$, where GH_k denotes the time-bounded garden-hose complexity, and D_k the k-round deterministic communication complexity. This result leads to strong lower bounds for the time bounded complexity of e.g. Equality, and to a time-hierarchy based on the pointer jumping problem.

Finally, we further investigate the power of randomness in the garden-hose model by considering private coin randomness ([4] consider only public coin randomness).

1.3 Organization

Most proofs are contained only in the full version of the paper, which is available on the arXiv.

2 Preliminaries

2.1 Definition of the Model

We now describe the garden-hose model in graph terminology. In a garden-hose protocol with s pipes there is a set V of s vertices plus one extra vertex, the *tap* t.

Given their inputs x, y Alice and Bob want to compute $f(x, y)$. Depending on x Alice connects some of the vertices in $V \cup \{t\}$ in pairs by adding edges $E_A(x)$ that form a matching among the vertices in $V \cup \{t\}$. Similarly Bob connects some of the vertices in V in pairs by adding edges $E_B(y)$ that form a matching in V.

Notice that after they have added the additional edges, a path starting from vertex t is formed in the graph $G = (V \cup \{t\}, E_A(x) \cup E_B(y))$. Since no vertex has degree larger than 2, this path is unique and ends at some vertex. We define the output of the game to be the parity of the length of the path starting at t. For instance, if the tap is not connected the path has length 0, and the output is 0. If the tap is connected to another vertex, and that vertex is the end of the path, then the path has length 1 and the output is 1 etc.

A garden-hose protocol for $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ is a mapping from $x \in \mathcal{X}$ to matchings among $V \cup \{t\}$ together with a mapping from $y \in \mathcal{Y}$ to matchings among V. The protocol computes $f(x, y)$ if for all x, y the path has even length iff $f(x, y) = 0$. The garden-hose complexity of f is the smallest s such that a garden-hose protocol of size s exists that computes f.

We note that one can form a matrix G_s that has rows labeled by all of Alice's matchings, and columns labeled by Bob's matchings, and contains the parity of the path lengths. A function f has garden-hose complexity s iff its communication matrix is a sub-matrix of G_s. G_s is called the *garden-hose matrix* for size s.

2.2 Communication Complexity, Formulae, Branching Programs

▶ **Definition 1.** Let $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$. In a communication complexity protocol two players Alice and Bob receive inputs x and y from $\{0, 1\}^n$. In the protocol players exchange messages in order to compute $f(x, y)$. Such a protocol is represented by a protocol tree, in which vertices, alternating by layer, belong to Alice or to Bob, edges are labeled with messages, and leaves either accept or reject. See [10] for more details. The communication matrix is the matrix containing $f(x, y)$ in row x and column y.

We say a protocol P correctly computes the function $f(x, y)$ if for all x, y the output of the protocol $P(x, y)$ is equal to $f(x, y)$. The communication complexity of a protocol is the maximum number of bits exchanged for all x, y.

The deterministic communication complexity $D(f)$ of a function f is the complexity of an optimal protocol that computes f.

▶ **Definition 2.** The non-deterministic communication complexity $N(f)$ of a Boolean function f is the length of the communication in an optimal two-player protocol in which Alice and Bob can make non-deterministic guesses, and there are three possible outputs `accept`, `reject`, `undecided`. For each x, y with $f(x, y) = 1$ there is a guess that will make the players accept but there is no guess that will make the players reject, and vice versa for inputs with $f(x, y) = 0$.

Note that the above is the two-sided version of non-deterministic communication complexity. It is well known [10] that $N(f) \leq D(f) \leq O(N^2(f))$, and that these inequalities are tight.

▶ **Definition 3.** In a public coin randomized protocol for f the players have access to a public source of random bits. For all inputs x, y it is required that the protocol gives the correct output with probability $1 - \epsilon$ for some $\epsilon < 1/2$. The public coin randomized communication complexity of f, $R_\epsilon^{pub}(f)$ is the complexity of the optimal public coin randomized protocol. Private coin protocols are defined analogously (players now have access only to private random bits), and their complexity is denoted by $R_\epsilon(f)$.

▶ **Definition 4.** The deterministic communication complexity of protocols with at most k messages exchanged, starting with Alice, is denoted by $D_k(f)$.

▶ **Definition 5.** In a simultaneous message passing protocol, both Alice and Bob send messages m_A, m_B to a referee. The referee, based on m_A, m_B, computes the output. The simultaneous communication complexity of a function f, $R^{||}(f)$, is the cost of the best simultaneous protocol that computes the function f using private randomness and error 1/3.

Next we define Boolean formulae.

▶ **Definition 6.** A Boolean formula is a Boolean circuit whose every node has fan-out 1 (except the output gate). A Boolean formula of depth d is then a tree of depth d. The nodes are labeled by gate functions from a family of allowed gate functions, e.g. the class of the 16 possible functions of the form $f : \{0,1\} \times \{0,1\} \to \{0,1\}$ in case the fan-in is restricted to 2. Another interesting class of gate functions is the class of all symmetric functions (of arbitrary fan-in). The *formula size* of a function f (relative to a class of gate functions) is the smallest number of leaves in a formula computing f.

Finally, we define branching programs. Our definition of permutation branching programs is extended in a slightly non-standard way.

▶ **Definition 7.** A branching program is a directed acyclic graph with one source node and two sink nodes (labeled with `accept` and `reject`). The source node has in-degree 0. The sink nodes have out-degree 0. All non-sink nodes are labeled by variables $x_i \in \{x_1, \cdots, x_n\}$ and have out-degree 2. The computation on an input x starts from the source node and depending on the value of x_i on a node either moves along the left outgoing edge or the right outgoing edge of that node. An input $x \in \{0,1\}^n$ is accepted iff the path defined by x in the branching program leads to the sink node labeled by `accept`. The length of the branching program is the maximum length of any path, and the size is the number of nodes.

A layered branching program of length l is a branching program where all non-sink nodes (except the source) are partitioned into l layers. All the nodes in the same layer query the same variable x_i, and all outgoing edges of the nodes in a layer go to the nodes in the next layer or directly to a sink. The width of a layered branching program is defined to be the maximum number of nodes in any layer of the program. We consider the starting node to be in layer 0 and the sink nodes to be in layer l.

A permutation branching program is a layered branching program, where each layer has the same number k of nodes, and if x_i is queried in layer i, then the edges labeled with 0 between layers i and $i+1$ form an injective mapping from $\{1, \ldots, k\}$ to $\{1, \ldots, k\} \cup \{\texttt{accept}, \texttt{reject}\}$ (and so do the the edges labeled with 0). Thus, for permutation branching programs if we fix the value of x_i, each node on level $i+1$ has in-degree at most 1.

We call a permutation branching program *strict* if there are no edges to `accept`/`reject` from internal layers. This is the original definition of permutation branching programs. Programs that are not strict are also referred to as *loose* for emphasis.

We denote by $PBP(f)$ the minimal size of a permutation branching program that computes f.

We note that simple functions like AND, OR can easily be computed by linear size loose permutation branching programs of width 2, something that is not possible for strict permutation branching programs [1].

3 Garden-Hose Protocols and Communication Complexity

3.1 Lower Bound via Non-deterministic Communication

In this section we show that non-deterministic communication complexity can be used to lower bound $GH(f)$. This bound is often better than the bound $GH(f) \geq \Omega(D_1(f)/\log(D_1(f)))$ shown in [4], which cannot be larger than $n/\log n$.

▶ **Theorem 8.** $GH(f) \geq N(f) - 1$.

The main idea is that a nondeterministic protocol that simulates the garden-hose game can choose the *set* of pipes that are used on a path used on inputs x, y instead of the path itself, reducing the complexity of the protocol. The set that is guessed may be a superset of the actually used pipes, introducing ambiguity. Nevertheless we can make sure that the additionally guessed pipes form cycles and are thus irrelevant.

As an application consider the function $IP(x,y) = \sum_{i=1}^{n}(x_i \cdot y_i) \textit{ mod } 2$. It is well known that $N(IP) \geq n+1$ [10], hence we get that $GH(IP) \geq n$. The same bound holds for Disjointness. These bounds improve on the previous $\Omega(n/\log n)$ bounds for these functions [4]. Furthermore note that the fooling set technique gives only bounds of size $\log^2 n$ for the complexity of IP (see [10]), so the technique previously used to get a linear lower bound for Equality fails for IP.

3.2 $GH(f)$ At Most The Size of a Protocol Tree for f

Buhrman et al. [4] show that any one way communication complexity protocol with complexity $D_1(f)$ can be converted to a garden-hose protocol with $2^{D_1(f)} + 1$ pipes. One-way communication complexity can be much larger than two-way communication [16].

▶ **Theorem 9.** *For any function f, the garden-hose complexity $GH(f)$ is upper bounded by the number of edges in a protocol tree for f.*

The construction is better than the previous one in [4] for problems for which one-way communication is far from the many-round communication complexity.

4 Relating Permutation Branching Programs and the Garden-Hose Model

▶ **Definition 10.** In a garden hose protocol a spilling-pipe on a player's side is a pipe such that water spills out of that pipe on the player's side during the computation for some input x, y.

We say a protocol has multiple spilling-pipes if there is more than one spilling-pipe on Alice's side or on Bob's side.

We now show a technical lemma that helps us compose garden-hose protocols without blowing up the size too much.

▶ **Lemma 11.** *A garden-hose protocol P for f with multiple spilling pipes can be converted to another garden-hose protocol P' for f that has only one spilling pipe on Alice's side and one spilling pipe on Bob's side. The size of P' is at most 3 times the size of P plus 1.*

Next we are going to show that it is possible to convert a (loose) permutation branching program into a garden-hose protocol with only a constant factor increase in size. We are stating a more general fact, namely that the inputs to the branching program we simulate can be functions (with small garden-hose complexity) instead of just variables. This allows us to use composition.

▶ **Lemma 12.** *$GH(g(f_1, f_2, \ldots, f_k)) = O(s \cdot \max(C_i)) + O(1)$, where $PBP(g) = s$ and $GH(f_i) = C_i$ and $f_i : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. The f_i do not necessarily have the same inputs x, y.*

A first corollary is the following fact already shown in [4]. Nonuniform Logspace is equal to the class of all languages recognizable by polynomial size families of branching programs. Since reversible Logspace equals deterministic Logspace [11], and a reversible Logspace machine (on a fixed input length) can be transformed into a polynomial size permutation branching program, we get the following.

▶ **Corollary 13.** *Logspace $\subseteq GH(poly(n))$. This holds for any partition of the variables among Alice and Bob.*

5 The Distributed Majority Function

In this section we investigate the complexity of the Distributed Majority function.

▶ **Definition 14.** Distributed Majority: $DMAJ(x, y) = 1$ iff $\sum_i^n (x_i \cdot y_i) \geq \frac{n}{2}$, where $x, y \in \{0,1\}^n$.

Buhrman et al. [4] have conjectured that the complexity of this function is quadratic, which is what is suggested by the naïve garden-hose protocol for the problem. The naïve protocol implicitly keeps one counter for i and one for the sum, leading to quadratic size. Here we describe a construction of a permutation branching program of size $O(n \cdot \log^3 n)$ for Majority, which can then be used to construct a garden-hose protocol for the Distributed Majority function. Note that the Majority function itself can be computed in the garden-hose model using $O(n)$ pipes (for any way to distribute inputs to Alice and Bob), since Alice can just communicate $\sum_i x_i$ to Bob. The advantage of using a permutation branching program to compute Majority is that by Lemma 12 we can then find a garden-hose protocol for the composition of Majority and the Boolean AND, which is the Distributed Majority function. Our construction of a permutation branching program adapts a branching program construction by Sinha and Thathachar [19].

▶ **Lemma 15.** *$PBP(MAJ) = O(n \cdot \log^3 n)$.*

We can now state our result about the composition of functions $f_1, \ldots, f_k$ with small garden-hose complexity via a Majority function.

▶ **Lemma 16.** *For $(f_1, f_2, \ldots, f_k)$, where each function f_i has garden-hose complexity $GH(f_i)$, $GH(MAJ(f_1, \ldots, f_k)) = O(\sum GH(f_i)) \cdot \log^3 k)$.*

The lemma immediately follows from combining Lemma 15 with Lemma 12. Considering $f_i = x_i \wedge y_i$ we get

▶ **Corollary 17.** *The garden-hose complexity of distributed Majority is $O(n \log^3 n)$.*

6 Composition and Connection to Formula Size

We wish to relate $GH(f)$ to the formula size of f. To do so we examine composition of garden-hose protocols by popular gate functions.

▶ **Theorem 18.** *For $(f_1, f_2, \ldots, f_k)$, where each function f_i has garden-hose complexity $GH(f_i)$*

- $GH(\bigvee f_i) = O(\sum GH(f_i))$.
- $GH(\bigwedge f_i) = O(\sum GH(f_i))$.
- $GH(\oplus f_i) = O(\sum GH(f_i))$.
- $GH(MAJ(f_i)) = O(\sum GH(f_i) \cdot \log^3 k)$.

This result follows from Lemma 16 and Lemma 12 combined with the trivial loose permutation branching programs for AND, OR, XOR.

We now turn to the simulation of Boolean formulae by garden-hose protocols. We use the simulation of formulae over the set of all fan-in 2 function by branching programs due to Giel [6].

▶ **Theorem 19.** *Let F be a formula for a Boolean function g on k inputs made of gates $\{\wedge, \vee, \oplus\}$ of arbitrary fan-in. If F has size s and $GH(f_i) \leq c$ for all i, then for all constants $\epsilon > 0$ we have $GH(g(f_1, f_2, \ldots, f_k)) \leq O(s^{1+\epsilon} \cdot c)$.*

Proof. Giel [6] shows the following simulation result:

▶ **Fact 1.** *Let $\epsilon > 0$ be any constant. Assume there is a formula with arbitrary fan-in 2 gates and size s for a Boolean function f. Then there is a layered branching program of size $O(s^{1+\epsilon})$ and width $O(1)$ that also computes f.*

By inspection of the proof it becomes clear that the constructed branching program is in fact a strict permutation branching program (an exponential increase in the width would yield this property in any case). The theorem follows by applying Lemma 12. ◀

▶ **Corollary 20.** *When the f_i's are single variables $GH(g) \leq O(s^{1+\epsilon})$ for all constants $\epsilon > 0$. Thus any lower bound on the garden-hose complexity of a function g yields a slightly smaller lower bound on formula-size (all gates of fan-in 2 allowed).*

The best lower bound of $\Omega(n^2/\log n)$ known for the size of formulae over the basis of all fan-in 2 gate function is due to Nečiporuk [12]. The Nečiporuk lower bound method (based on counting subfunctions) can also be used to give the best general branching program lower bound of $\Omega(n^2/\log^2 n)$ (see [20]).

Due to the above any lower bound larger than $\Omega(n^{2+\epsilon})$ for the garden-hose model would immediately give lower bounds of almost the same magnitude for formula size and permutation branching program size. Proving super-quadratic lower bounds in these models is a long-standing open problem.

Due to the fact that we have small permutation branching programs for Majority, we can even simulate a more general class of formulae involving a limited number of Majority gates.

▶ **Theorem 21.** *Let F be a formula for a Boolean function g on n inputs made of gates $\{\wedge, \vee, \oplus\}$ of arbitrary fan-in. Additionally, on any path from the root to the leaves there may be up to $O(1)$ Majority gates. If F has size s, then for all constants $\epsilon > 0$ we have $GH(g) \leq O(s^{1+\epsilon})$.*

Proof. Proceeding in reverse topological order we can replace all sub-formulae below a Majority gate by garden-hose protocols with Theorem 19, increasing the size of the sub-formula. Then we can apply Lemma 16 to replace the sub-formula including the Majority gate by a garden-hose protocol. If the size of the formula below the Majority gate is $\tilde{s}$, then the garden-hose size is $O(\tilde{s}^{1+\epsilon'})$, where the poly-logarithmic factor of Lemma 16 is hidden in the polynomial increase. Since every path from root to leaf has at most $c = O(1)$ Majority gates, and we may choose the ϵ' in Theorem 19 to be smaller than ϵ/c, we get our result. ◀

6.1 The Nečiporuk Bound with Arbitrary Symmetric Gates

Since garden-hose protocols can even simulate formulae containing some arbitrary fan-in Majority gates, the question arises whether one can hope for super-linear lower bounds at all. Maybe it is hard to show super-linear lower bounds for formulae having Majority gates? Note that very small formulae for the Majority function itself are not known (the currently best construction yields formulae of size $O(n^{3.03})$ [18]), hence we cannot argue that Majority gates do not add power to the model. In this subsection we sketch the simple observation that the Nečiporuk method [12] can be used to give good lower bounds for formulae made of *arbitrary symmetric gates of any fan-in*. Hence there is no obstacle to near-quadratic lower bounds from the formula size connection we have shown. We stress that nevertheless we do not have any super-linear lower bounds for the garden-hose model.

We employ the communication complexity notation for the Nečiporuk bound from [8].

▶ **Theorem 22.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and $B_1, \ldots, B_k$ a partition of the input bits of f. Denote by $D_j(f)$ the deterministic one-way communication complexity of f, when Alice receives all inputs except those in B_j, and Bob the inputs in B_j. Then the size (number of leaves) of any formula consisting of arbitrary symmetric Boolean gates is at least $\sum D_j(f)/\log n$.*

The theorem is as good as the usual Nečiporuk bound except for the log-factor, and can hence be used to show lower bounds of up to $\Omega(n^2/\log^2 n)$ on the formula size of explicit functions like IndirectStorageAccess [20].

7 Time Bounded Garden-Hose Protocols

We now define a notion of time in garden-hose complexity.

▶ **Definition 23.** Given a garden-hose protocol P for computing function f, and an input x, y we refer to the pipes that carry water in P on x, y as the wet pipes. Let T_P denote the maximum number of wet pipes for any input (x, y) in P.

The number of wet pipes on input x, y is equal to the length of the path the water takes and thus corresponds to the time the computation takes. Thus it makes sense to investigate protocols which have bounded time T_P. Furthermore, the question is whether it is possible to simultaneously optimize T_P and the number of pipes used.

▶ **Definition 24.** We define $GH_k(f)$ to be the complexity of an optimal garden-hose protocol P for computing f where for any input (x, y) we have that T_P is bounded by k.

As an example consider the Equality function (test whether $x = y$). The straightforward protocol that compares bit after bit has cost $3n$ but needs time $2n$ in the worst case. On the other hand one can easily obtain a protocol with time 2, that has cost $O(2^n)$: use 2^n pipes to communicate x to Bob. We have the following general lower bound.

▶ **Theorem 25.** *For all Boolean functions f we have $GH_k(f) = \Omega(2^{D_k(f)/k})$, where $D_k(f)$ is the deterministic communication complexity of f with at most k rounds (Alice starting).*

Proof. We rewrite the claim as $D_k(f) = O(k \cdot \log GH_k(f))$.

Let P' be the garden-hose protocol for f that achieves complexity $GH_k(f)$ for f. The deterministic k-round communication protocol for f simulates P' by simply following the flow of the water. In each round Alice or Bob (alternatingly) send the name of the pipe used at that time by P'. ◀

Thus for Equality we have for instance that $GH_{\sqrt{n}}(Equality) = \Omega(2^{\sqrt{n}})$. There is an almost matching upper bound of $GH_{\sqrt{n}}(Equality) = O(2^{\sqrt{n}} \cdot \sqrt{n})$ by using $\sqrt{n}$ blocks of $2^{\sqrt{n}}$ pipes to communicate blocks of $\sqrt{n}$ bits each.

We can easily deduce a time-cost tradeoff from the above: For Equality the product of time and cost is at least $\Omega(n^2/\log n)$, because for time $T < o(n/\log n)$ we get a super-linear bound on the size, whereas for larger T we can use that the size is always at least n.

7.1 A Time-Size Hierarchy

The Pointer Jumping Function is well-studied in communication complexity. We describe a slight restriction of the problem in which the inputs are permutations of $\{1, \dots, n\}$.

▶ **Definition 26.** Let U and V be two disjoint sets of vertices such that $|U| = |V| = n$.

Let $F_A = \{f_A | f_A : U \to V$ and f_A is bijective$\}$ and $F_B = \{f_B | f_B : V \to U$ and f_B is bijective$\}$. For a pair of functions $f_A \in F_A$ and $f_B \in F_B$ define $f(v) = \begin{cases} f_A(v) & \text{if } v \in U \\ f_B(v) & \text{if } v \in V. \end{cases}$

Then $f_0(v) = v$ and $f_k(v) = f(f_{k-1}(v))$.

Finally, the pointer jumping function $PJ_k : F_A \times F_B \to \{0, 1\}$ is defined to be the XOR of all bits in the binary name of $f_k(v_0)$, where v_0 is a fixed vertex in U.

Round-communication hierarchies for PJ_k or related functions are investigated in [15]. Here we observe that PJ_k gives a time-size hierarchy in the garden-hose model. For simplicity we only consider the case where Alice starts.

▶ **Theorem 27.** **1.** *PJ_k can be computed by a garden-hose protocol with time k and size kn.*
2. *Any garden-hose protocol for PJ_k that uses time at most $k - 1$ has size $2^{\Omega(n/k)}$ for all $k \le n/(100 \log n)$.*

We note that slightly weaker lower bounds hold for the randomized setting.

8 Randomized Garden-Hose Protocols

We now bring randomness into the picture and investigate its power in the garden-hose model. Buhrman et al [4] have already considered protocols with public randomness. In this section we are mainly interested in the power of private randomness.

▶ **Definition 28.** Let $RGH^{pub}(f)$ denote the minimum complexity of a garden-hose protocol for computing f, where the players have access to public randomness, and the output is correct with probability 2/3 (over the randomness). Similarly, we can define $RGH^{pri}(f)$, the cost of garden-hose protocols with access to private randomness.

By standard fingerprinting ideas [10] we can observe the following.

▶ **Claim 1.** *$RGH^{pub}(Equality) = O(1)$.*

▶ **Claim 2.** *$RGH^{pri}(Equality) = O(n)$, and this is achieved by a constant time protocol.*

Proof. The second claim follows from Newman's theorem [13] showing that any public coin protocol with communication cost c can be converted into a private coin protocol with communication cost $c + \log n + O(1)$ bits on inputs of length n together with the standard public coin protocol for Equality, and the protocol tree simulation of Theorem 9. ◀

Of course we already know that even the deterministic complexity of Equality is $O(n)$, hence the only thing achieved by the above protocol is the reduction in time complexity. Note that due to our result of the previous section computing Equality deterministically in constant time needs exponentially many pipes.

Buhrman et al. [4] have shown how to de-randomize a public coin protocols at the cost of increasing size by a factor of $O(n)$, so the factor n in the separation between public coin and deterministic protocols above is the best that can be achieved. This raises the question whether private coin protocols can ever be more efficient in size than the optimal deterministic protocol. We now show that there are no very efficient private coin protocols for Equality.

▶ **Claim 3.** *$RGH^{pri}(Equality) = \Omega(\sqrt{n}/\log n)$*

Proof. To prove this we first note that $RGH^{pri}(f) = \Omega(R^{||}(f)/\log R^{||}(f))$, where $R^{||}(f)$ is the cost of randomized private coin simultaneous message protocols for f (Alice and Bob can send their connections to the referee). Hence, $RGH^{pri}(f) = \Omega(R^{||pri}(f)/\log R^{||pri}(f))$, but Newman and Szegedy [14] show that $RGH^{pri}(Equality) = \Omega(\sqrt{n})$. ◀

9 Open Problems

- We show that getting lower bounds on $GH(f)$ larger than $\Omega(n^{2+\epsilon})$ will be hard. But we know of no obstacles to proving super-linear lower bounds.
- Possible candidates for quadratic lower bounds could be the Disjointness function with set size n and universe size n^2, and the IndirectStorageAccess function.
- Consider the garden-hose matrix G_s as a communication matrix. How many distinct rows does G_s have? What is the deterministic communication complexity of G_s? The best upper bound is $O(s \log s)$, and the lower bound is $\Omega(s)$. An improved lower bound would give a problem, for which $D(f)$ is larger than $GH(f)$.
- We have proved $RGH^{pri}(Equality) = \Omega(\sqrt{n}/\log n)$. Is it true that $RGH^{pri}(Equality) = \Theta(n)$? Is there any problem where $RGH^{pri}(f)$ is smaller than $GH(f)$?
- It would be interesting to investigate the relation between the garden-hose model and memoryless communication complexity, i.e., a model in which Alice and Bob must send messages depending on their input and the message just received only. The garden-hose model is memoryless, but also reversible.

Acknowledgement. We thank an anonymous referee for pointing out a mistake in an earlier version of this paper.

References

1 D. A. Barrington. Width-3 permutation branching programs, 1985. Technical report, MIT/LCS/TM-293.

2 P. Beame, M. Tompa, and P. Yan. Communication-space tradeoffs for unrestricted protocols. *SIAM Journal on Computing*, 23(3):652–661, 1994. Earlier version in FOCS'90.

3 Joshua Brody, Shiteng Chen, Periklis A. Papakonstantinou, Hao Song, and Xiaoming Sun. Space-bounded communication complexity. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 159–172, 2013.

4 Harry Buhrman, Serge Fehr, Christian Schaffner, and Florian Speelman. The garden-hose model. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 145–158. ACM, 2013.

5 Well Y. Chiu, Mario Szegedy, Chengu Wang, and Yixin Xu. The garden hose complexity for the equality function. *arXiv:1312.7222*, 2013.

6 O. Giel. Branching program size is almost linear in formula size. *Journal of Computer and System Sciences*, 63(2):222–235, 2001.

7 H. Klauck. Quantum and classical communication-space tradeoffs from rectangle bounds. In *Proceedings of FSTTCS*, 2004.

8 H. Klauck. One-Way Communication Complexity and the Nečiporuk Lower Bound on Formula Size. *SIAM J. Comput.*, 37(2):552–583, 2007.

9 H. Klauck, R. Špalek, and R. de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM Journal on Computing*, 36(5):1472–1493, 2007. Earlier version in FOCS'04. quant-ph/0402123.

10 Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, 1997.

11 K. J. Lange, P. McKenzie, and A. Tapp. Reversible space equals deterministic space. *Journal of Computer and System Sciences*, 2(60):354–367, 2000.

12 E. I. Nečiporuk. A boolean function. *Soviet Mathematics Doklady*, 7(S 999), 1966.

13 I. Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2):67–71, 1991.

14 Ilan Newman and Mario Szegedy. Public vs. private coin flips in one round communication games (extended abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC'96, pages 561–570, 1996.

15 Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. *SIAM J. Comput.*, 22(1):211–219, February 1993.

16 C. H. Papadimitriou and M. Sipser. Communication complexity. *Journal of Computer and System Sciences*, 28(2):260–269, 1984. Earlier version in STOC'82.

17 P. Papakonstantinou, D. Scheder, and H. Song. Overlays and limited memory communication mode(l)s. In *Proc. of the 29th Conference on Computational Complexity*, 2014.

18 I. S. Sergeev. Upper bounds for the formula size of symmetric boolean functions. *Russian Mathematics, Iz. VUZ*, 58(5):30–42, 2014.

19 Rakesh Kumar Sinha and Jayram S Thathachar. Efficient oblivious branching programs for threshold and mod functions. *Journal of Computer and System Sciences*, 55(3):373–384, 1997.

20 I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner Series in Computer Science, 1987.

Homomorphism Polynomials Complete for VP*

Arnaud Durand[1], Meena Mahajan[2], Guillaume Malod[1], Nicolas de Rugy-Altherre[1], and Nitin Saurabh[2]

1 **Univ Paris Diderot, Sorbonne Paris Cité, IMJ-PRG, UMR 7586 CNRS, Sorbonne Université, UPMC Univ Paris 06, F-75013, Paris, France**
{durand,malod,nderugy}@math.univ-paris-diderot.fr

2 **The Institute of Mathematical Sciences, CIT Campus, Chennai 600113, India**
{meena,nitin}@imsc.res.in

Abstract

The VP versus VNP question, introduced by Valiant, is probably the most important open question in algebraic complexity theory. Thanks to completeness results, a variant of this question, VBP versus VNP, can be succinctly restated as asking whether the permanent of a generic matrix can be written as a determinant of a matrix of polynomially bounded size. Strikingly, this restatement does not mention any notion of computational model. To get a similar restatement for the original and more fundamental question, and also to better understand the class itself, we need a complete polynomial for VP. Ad hoc constructions yielding complete polynomials were known, but not natural examples in the vein of the determinant. We give here several variants of natural complete polynomials for VP, based on the notion of graph homomorphism polynomials.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases algebraic complexity, graph homomorphism, polynomials, VP, VNP, completeness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.493

1 Introduction

One of the most important open questions in algebraic complexity theory is to decide whether the classes VP and VNP are distinct. These classes, first defined by Valiant in [12, 13], are the algebraic analogues of the Boolean complexity classes P and NP, and separating them is essential for separating P from NP (at least non-uniformly and assuming the generalised Riemann Hypothesis, over the field $\mathbb{C}$, [3]). Valiant established that the family of polynomials computing the permanent is complete for VNP under a suitable notion of reduction which can be thought of as a very strong form of polynomial-size reduction. The leading open question of VP versus VNP is often phrased as the permanent versus the determinant, as the determinant is complete for VP. However, the hardness of the determinant for VP is under the more powerful quasi-polynomial-size reductions. Under polynomial reductions, the determinant is complete for the possibly smaller class VBP. This naturally raises the question of finding polynomials which are complete for VP under polynomial-size reductions. Ad hoc families of generic polynomials can be constructed that are VP-complete, but, surprisingly, there are no known natural polynomial families that are VP-complete. Since complete problems characterise complexity classes, the existence of natural complete problems lends

* This work was supported by IFCPAR/CEFIPRA Project 4702-1(A).

added legitimacy to the study of a class. The determinant and the permanent make the classes VBP, VNP interesting; analogously, what characterises VP?

Our results and techniques

In this paper, we provide the first instance of natural families of polynomials that (1) are defined independently of the circuit definition of VP, and (2) are VP-complete. The families we consider are families of homomorphism polynomials. Formal definitions appear in Section 2, but here is a brief description. For graphs G and H, a homomorphism from "source graph" G to "target graph" H is a map from $V(G)$ to $V(H)$ that preserves edges. If G and H are directed, a directed homomorphism must preserve directed edges. Additionally, if the vertices of G and H are coloured, a coloured homomorphism must also preserve colours. Placing distinct variables on the vertices (X variables) and edges (Y variables) of the target graph H, we can associate with each homomorphism from G to H a monomial built using these variables. The homomorphism polynomial associated with G and H is the sum of all such monomials. Various variants can be obtained by (1) summing only over homomorphisms of a certain type $\mathcal{H}$, e. g., directed, coloured, injective,... (2) setting non-negative weights α on the vertices of G and using these weights while defining the monomial associated with a homomorphism. Thus the general form of a homomorphism polynomial is $f_{G,H,\alpha,\mathcal{H}}(X, Y)$. We show that over fields of characteristic zero, with respect to constant-depth oracle reductions, the following natural settings, in order of increasing generality, give rise to VP-complete families (Theorem 19):

1. G is a balanced alternately-binary-unary tree with n leaves, with a marker gadget added to the root, and with edge directions chosen in a specific way; H is the complete directed graph on n^6 nodes; α is 1 everywhere; $\mathcal{H}$ is the set of directed homomorphisms.
2. G is an undirected balanced alternately-binary-unary tree with n leaves; H is the complete undirected graph on n^6 nodes; α is 1 everywhere; the vertices are coloured with 5 colours in a specific way; $\mathcal{H}$ is the set of coloured homomorphisms.
3. G is a balanced binary tree with n leaves; H is a complete graph on n^6 nodes; α is 1 for every right child in G and 0 elsewhere; $\mathcal{H}$ is the set of all homomorphisms from G to H.

There seems to be a trade-off between the ease of describing the source and target graphs and the use of weights α. The first family above does not use weights (α is 1 everywhere), but G needs a marker gadget on a naturally defined graph. The second family also does not use weights (α is 1 everywhere), but the colouring of H is described with reference to previously known universal circuits. The third family has very natural source and target graphs, but requires non-trivial α. Ideally, we should be able to show VP-completeness with G and H as in the third family and with trivial weights as in the first two families; our hardness proofs fall short of this. Note however that the weights we use are 0-1 valued. Such 0-1 weights are commonly used in the literature, see, e. g., [2].

A crucial ingredient in our hardness proofs is the fact that VP circuits can be depth-reduced [14] and made multiplicatively disjoint [8] so that all parse trees are isomorphic to balanced binary trees. Another crucial ingredient is that homogeneous components of a polynomial p can be computed in constant depth and polynomial size with oracle gates for p. The hardness proofs illustrate how the monomials in the generic VP-complete polynomial can be put in correspondence with a carefully chosen homogeneous component of the homomorphism polynomial (equivalently, with monomials associated with homomorphisms and satisfying some degree constraints in certain variables). Extracting the homogeneous component is what necessitates an oracle-reduction (constant depth suffices) for hardness. The coloured homomorphism polynomial is however hard even with respect to projections, the stricter form of polynomial-size reductions which is more common in this setting.

For all the above families, membership in VP is shown in a uniform way by showing that a more general homomorphism polynomial, where we additionally have a set of variables Z for each pair of nodes $V(G) \times V(H)$, is in VP, and that the above variants can be obtained from this general polynomial through projections. The generalisation allows us to partition the terms corresponding to $\mathcal{H}$ into groups based on where the root of G is mapped, factorise the sums within each group, and recurse. A crucial ingredient here is the powerful Baur-Strassen Lemma 3 ([1]) which says that for a polynomial p computed by a size s circuit, p and all its first-order derivatives can be simultaneously computed in size $O(s)$.

We also show that when G is a path (instead of a balanced binary tree), the homomorphism polynomial family is complete for VBP. On the other hand, using the generalised version with Z variables, and letting G, H be complete graphs, we get completeness for VNP.

Previous related results

As mentioned earlier, very little was previously known about VP-completeness. In [3], Bürgisser showed that a generic polynomial family constructed recursively while controlling the degree is complete for VP (Bürgisser showed something even more general; completeness for relativised VP). The construction directly follows a topological sort of a generic VP circuit. In [10] (see also [11]), Raz used the depth-reduction of [14] to show that a family of "universal circuits" is VP-complete; any VP computation can be embedded into it by appropriately setting the variables. Both these VP-complete families are thus directly obtained using the circuit definition / characterization of VP. In [9], Mengel described a way of associating polynomials with constraint satisfaction programs CSPs, and showed that for CSPs where all constraints are binary and the underlying constraint graph is a tree, these polynomials are in VP. Further, for each VP-polynomial, there is such a CSP giving rise to the same polynomial. This means that for the CSP corresponding to the generic VP polynomial or universal circuit, the associated polynomial is VP-complete. The unsatisfactory element here is that to describe the complete polynomial, one again has to fall back to the circuit definition of VP. Similarly, in [4], it is shown that tensor formulas can be computed in VP and can compute all polynomials in VP. Again, to put our hands on a specific VP-complete tensor formula, we need to fall back to the circuit characterisation of VP.

For VBP, on the other hand, there are natural known complete problems, most notably the determinant and iterated matrix multiplication.

A somewhat different homomorphism polynomial was studied in [5]; for a graph H, the monomials of the polynomial f_n^H encode the distinct graphs of size n that are homomorphic to H. The dichotomy result established there gives completeness for VNP or membership in Valiant's analogue of AC^0; it does not capture VP.

Finally, a considerable number of works have been done during the last years on the related subject of counting graph homomorphisms (but mostly in the non uniform settings – i. e., when the target graph is fixed – see [7]) or counting models of CSP and conjunctive queries with connections to VP-completeness (see [6]).

Organization of this paper

In Section 2, basic definitions and notations and previous results used are stated. In Section 3 we describe the hardness of various homomorphism polynomials for VP. Membership in VP is established in Section 4. Completeness for VBP and VNP is discussed in Section 5. Due to space limitations, detailed proofs had to be omitted.

2 Preliminaries and Notation

An arithmetic circuit is a directed acyclic graph with leaves labeled by variables or field elements, internal nodes (called gates) labeled by one of the field operations $+$ and $\times$, and designated output gates at which specific polynomials are computed in the obvious way. If every node has fan-out at most 1 (only one successor), then the circuit is a formula (the underlying graph is a tree). If at every node labeled $\times$, the subcircuits rooted at the children of the node are disjoint, then the circuit is said to be multiplicatively disjoint. Notions of size and depth are similar to that of classical Boolean circuits. For more details about arithmetic circuits, see for instance [11].

A family of polynomials $\{f_n(x_1, \ldots, x_{t(n)})\}$ is p-bounded if $t(n)$ and degree(f_n) are $n^{O(1)}$. A p-bounded family $\{f_n\}$ is in VP if a circuit family $\{C_n\}$ of size $s(n) \in n^{O(1)}$ computes it.

▶ **Proposition 1** ([14, 8]). *If $\{f_n\}$ is in* VP*, then $\{f_n\}$ can be computed by polynomial-size circuits of depth $O(\log n)$ where each $\times$ gate has fan-in at most 2. Furthermore, the circuits are multiplicatively disjoint.*

We say that $\{f_n\}$ is a p-projection of $\{g_n\}$ if there is an $m(n) \in n^{O(1)}$ such that each f_n can be obtained from $g_{m(n)}$ by setting each of the variables in $g_{m(n)}$ to a variable of f_n or to a field element.

A **constant-depth c-reduction** from $\{f_n\}$ to $\{g_n\}$, denoted $f \leq_c g$, is a polynomial-size constant-depth circuit family with $+$ and $\times$ gates and oracle gates for g, that computes f. (This is akin to AC^0-Turing reductions in the Boolean world.)

A family $\{D_n\}$ of **universal circuits** computing a polynomial family $\{p_n\}$ is described in [10, 11]. These circuits are universal in the sense that that every polynomial $f(X_1, \ldots, X_n)$ of degree d, computed by a circuit of size s, can be computed by a circuit Ψ such that the underlying graph of Ψ is the same as the graph of D_m, for $m \in \text{poly}(n, s, d)$. (In fact, f_n can be obtained as a projection of p_m.) With minor modifications to $\{D_n\}$ (simple padding with dummy gates, followed by the multiplicative disjointness transformation from [8]), we can show that there is a universal circuit family $\{C_n\}$ in the normal form described below:

▶ **Definition 2** (Normal Form Universal Circuits). A universal circuit $\{C_n\}$ in normal form is a circuit with the following structure:

- It is a layered and semi-unbounded circuit, where $\times$ gates have fan-in 2, whereas $+$ gates are unbounded.
- Gates are alternating, namely every child of a $\times$ gate is a $+$ gate and vice versa. Without loss of generality, the root is a $\times$ gate.
- All the input gates have fan-out 1 and they are at the same level, i. e., all paths from the root of the circuit to an input gate have the same length.
- C_n is a multiplicatively disjoint circuit.
- Input gates are labeled by distinct variables. In particular, there are no input gates labeled by a constant.
- Depth $(C_n) = 2k(n) = 2c\lceil \log n \rceil$; number of variables $(\bar{x}) = v_n$; and size $(C_n) = s_n$, which is polynomial in n.
- The degree of the polynomial computed by the universal circuit is n.

We will identify the directed graph of the circuit, where each edge e is labeled by a new variable X_e, by the circuit itself. Let $(\mathsf{f}_{C_n}(\bar{x}))_n$ be the polynomial family computed by the universal circuit family in normal form.

▶ **Lemma 3** ([1]). *Let $L(p_1, p_2, \ldots, p_k)$ denote the size of a smallest circuit computing the polynomials p_i at k of its nodes. For any $f \in \mathbb{F}[\bar{x}]$,*

$$L\left(f, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right) \leq 3L(f).$$

The coefficient of a particular monomial in a polynomial can be extracted as described below. It appears to be folklore, and was noted in [3]; a version appears in [5] (Lemma 2).

▶ **Lemma 4** (Folklore). *Let F be any field of characteristic zero.*

1. *Let p be a polynomial in $F(\bar{W})$, with total degree at most D. Let m be any monomial, with k distinct variables appearing in it. The coefficient of m in p can be computed by a $O(k)$-depth circuit of size $O(Dk)$ with oracle gates for p.*
2. *Let p be a polynomial in $F(\bar{X}, \bar{W})$, with $|\bar{W}| = n$ and total degree in $\bar{W}$ at most D. Let p_d denote the component of p of total degree in $\bar{W}$ exactly d. Then p_d can be computed by a constant depth circuit of size $O(Dn)$ with $O(D)$ oracle gates for p.*

We use (u, v) to denote an undirected edge between u and v, and $\langle u, v\rangle$ to denote a directed edge from u to v.

▶ **Definition 5** (Homomorphisms). Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be two undirected graphs. A homomorphism from G to H is a mapping $\phi : V(G) \to V(H)$ such that the image of an edge is an edge; i.e., for all $(u, v) \in E(G)$, $(\phi(u), \phi(v)) \in E(H)$.

If G, H are directed graphs, then a homomorphism only needs to satisfy for all $\langle u, v\rangle \in E(G)$, at least one of $\langle \phi(u), \phi(v)\rangle, \langle \phi(v), \phi(u)\rangle$ is in $E(H)$. But a directed homomorphism must satisfy for all $\langle u, v\rangle \in E(G)$, $\langle \phi(u), \phi(v)\rangle \in E(H)$.

If c_G, c_H are functions assigning colours to $V(G)$ and $V(H)$, then a coloured homomorphism must also satisfy, for all $u \in V(G)$, $c_G(u) = c_H(\phi(u))$.

▶ **Definition 6** (Homomorphism polynomials (see, e.g., [2])). Let G and H be undirected graphs; the definitions for the directed case are analogous. Consider the set of variables $X \cup Y$ where $X = \{X_u | u \in V(H)\}$ and $Y = \{Y_{uv} | (u, v) \in E(H)\}$. Let $\alpha : V(G) \mapsto \mathbb{N}$ be a labeling of vertices of G by non-negative integers. For each homomorphism ϕ from G to H we associate the monomial

$$mon(\phi) \triangleq \left(\prod_{u \in V(G)} X_{\phi(u)}^{\alpha(u)}\right)\left(\prod_{(u,v) \in E(G)} Y_{\phi(u),\phi(v)}\right).$$

Let $\mathcal{H}$ be a set of homomorphisms from G to H. The homomorphism polynomial $f_{G,H,\alpha,\mathcal{H}}$ is defined as follows:

$$f_{G,H,\alpha,\mathcal{H}}(X, Y) = \sum_{\phi \in \mathcal{H}} mon(\phi) = \sum_{\phi \in \mathcal{H}} \left(\prod_{u \in V(G)} X_{\phi(u)}^{\alpha(u)}\right)\left(\prod_{(u,v) \in E(G)} Y_{\phi(u),\phi(v)}\right).$$

Some sets of homomorphisms we consider are **InjDirHom**: injective directed homomorphisms, **InjHom**: injective homomorphisms, **DirHom**: directed homomorphisms, **ColHom**: coloured homomorphisms, **Hom**: all homomorphisms.

▶ **Definition 7** (Parse trees (see, e.g., [8])). The set of parse trees of a circuit C is defined by induction on its size:

- If C is of size 1, it has only one parse tree, itself.

- If the output gate of C is a $\times$ gate whose children are the gates α and β, the parse trees of C are obtained by taking a parse tree of C_α, a parse tree of a disjoint copy of C_β and the edges from α and β to the output gate.
- If the output of C is a $+$ gate, the parse trees of C are obtained by taking a parse tree of a subcircuit rooted at one of the children and the edge from the (chosen) child to the output gate.

Each parse tree T is associated with a monomial by computing the product of the values of the input gates. We denote this value by $mon(\mathsf{T})$.

▶ **Lemma 8** ([8]). *If C is a circuit computing a polynomial f, then $f(\bar{x}) = \sum_{\mathsf{T}} \mathrm{mon}(\mathsf{T})$, where the sum is over the set of parse trees, T, of C.*

▶ **Proposition 9** ([8]). *A circuit C is multiplicatively disjoint if and only if any parse tree of C is a subgraph of C. Furthermore, a subgraph T of C is a parse tree if the following conditions are met:*
- *T contains the output gate of C.*
- *If α is a multiplication gate in T having gates β and γ as children in C, then the edges $\langle\beta,\alpha\rangle$ and $\langle\gamma,\alpha\rangle$ also appear in T.*
- *If α is an addition gate in T, it has only one child in T.*
- *Only edges and gates obtained in this way belong to T.*

3 Lower Bounds: VP-hardness

Here we study the question of whether all families of polynomials in VP can be computed by homomorphism polynomials. Instantiating G, H and α to our liking we obtain a variety of homomorphism polynomials that are VP-hard. We describe them in increasing order of generalisation.

▶ **Definition 10.** Let AT_k be a directed balanced alternately-binary-unary tree with k leaves. Vertices on an odd layer have exactly two incoming edges whereas vertices on an even layer have exactly one incoming edge. The first layer has only one vertex called root, and the edges are directed from leaves towards the root.

▶ **Lemma 11.** *The parse trees of C_n, the universal circuit in normal form, are subgraphs of C_n and are isomorphic to AT_n.*

This observation suggests a way to capture monomial computations of the universal circuit via homomorphisms from AT_k into C_n.

Injective Directed Homomorphism

▶ **Proposition 12.** *Consider the homomorphism polynomial where*
- *$G := \mathsf{AT}_m$.*
- *H is the directed graph corresponding to the universal circuit in normal form C_m.*
- *$\mathcal{H} :=$ set of injective directed homomorphisms from G to H.*
- *α is 1 everywhere.*

Then, the family $(f_{\mathsf{AT}_m,H,\alpha,\textbf{\textit{InjDirHom}}}(\bar{X},\bar{Y}))_m$, where $m \in \mathbb{N}$, is VP-hard for projections.

Proof Sketch. We want to express the universal polynomial through a projection. We set all X variables at leaves to the corresponding variables in C_m, and all other X variables and all Y variables to 1. The idea is to show that elements in **InjDirHom** are in bijection with parse trees of C_m, and compute the same monomials. It is easy to see that for every parse tree of C_m, there is a $\phi \in$ **InjDirHom** with exactly this image. On the other hand, every $\phi \in$ **InjDirHom** must map the directed paths of length $2\log m$ in G to directed paths in H; this fact can be used to show that its image is a parse tree of C_m. ◀

▶ Remark. The hardness proof above will work even if H is the complete directed graph on poly(m) nodes. In the projection, we can set the $\bar{Y}$ variables to values in $\{0, 1\}$ such that the edges with variables set to 1 together form the underlying graph of C_n.

If we follow the proof of the previous proposition and look at the image of a given homomorphism in layers, we notice that "direction"-respecting homomorphisms basically ensured that we never fold back (in the image). In particular, the mapping respect layers. Furthermore "injectivity" helped ensure that vertices within a layer are mapped distinctly. This raises an intriguing question: can we eliminate either assumption (*direction* or *injectivity*) and still prove VP-hardness? We answer this question positively, albeit under a stronger notion of reduction.

Injective Homomorphisms

Let AT_k^u be defined as the alternately-binary-unary tree AT_k, but with no directions on edges.

▶ **Proposition 13.** *Consider the homomorphism polynomial where*

- $G := \mathsf{AT}_m^u$.
- *H is a complete graph (undirected) on poly(m), say m^6, nodes.*
- *$\mathcal{H}$:= set of injective homomorphisms from G to H.*
- *α is 1 everywhere.*

Then, the family $(f_{\mathsf{AT}_m^u, H, \alpha, \textbf{\textit{InjHom}}}(\bar{X}, \bar{Y}))_m$ is VP-hard for constant-depth c-reductions.

Proof Sketch. Again, we want to express the universal polynomial. Setting some Y variables to 0 values allows us to pick out C_m from H. To enforce directedness of the injective homomorphisms, we assign a special variable r on the edges emerging from the root, and a special variable ℓ on edges reaching the leaves. (The remaining Y variables are set to 1; the X variables are set as in Proposition 12.) Now the coefficient of $\ell^m r^2$ in f extracts exactly the contribution of injective directed homomorphisms, and this, by Proposition 12, is the universal polynomial. The desired coefficient can be extracted by a constant-depth c-reduction, as described in Lemma 4. ◀

Directed Homomorphisms

Consider the directed alternately-binary-unary-tree AT_k. For every vertex in an odd layer there are two incoming edges. Flip the direction of the right edge for every such vertex. Note that the edges coming into the unary vertices at even layers are unchanged. Also connect a path $t_1 \to t_2 \to \cdots \to t_s$ to the root by adding an edge $\langle t_s, root \rangle$. The vertices $t_1, \ldots, t_s$ are new vertices. Denote this modified alternately-binary-unary-tree by $\mathsf{AT}_{k,s}^d$.

▶ **Theorem 14.** *Consider the homomorphism polynomial where*

- $G := \mathsf{AT}_{m,s}^d$ *for sufficiently large s in poly(m), say $s = m^7$.*
- *H is a complete directed graph on poly(m), say m^6, nodes.*
- *$\mathcal{H}$:= set of directed homomorphisms from G to H.*
- *α is 1 everywhere.*

Then, the family $(f_{\mathsf{AT}_{m,s}^d, H, \alpha, \textbf{\textit{DirHom}}}(\bar{X}, \bar{Y}))_m$ is VP-hard for constant-depth c-reductions.

Proof Sketch. To compute the universal polynomial, we set some Y variables to 0 to pick out from H the circuit C_m with a tail at the root. We assign special variables r and t on the first and last edge of the tail, and variable ℓ on the edges entering leaves of C_m. The idea is to show that homomorphism monomials with degree exactly 1 in r and in t and degree exactly m in ℓ are in bijection with parse trees of C_m (and compute the same corresponding monomials). The length of the tail and the degree in r and t, ensure that any directed homomorphism maps the tail in G to the tail attached to C_m, so the root of the copy of AT_m inside G is mapped to the root of C_m. The degree constraint in ℓ ensures that the leaves of AT_m are mapped to the leaves of C_m, thus preserving layers. An inductive argument based on layer numbers, beginning from the root, and using the multiplicative disjointness of C_m, shows that the homomorphisms must also be injective. This gives the bijection. ◀

Coloured Homomorphisms

In all the above hardness proofs we restricted the set of homomorphisms to be *direction-respecting*, or *injective*, or both. Here we show another restriction, called *colour-respecting*, that gives a VP-hard polynomial. Recall that a homomorphism from a coloured graph to another coloured graph is *colour-respecting* if it preserves the colour class of vertices.

Consider the following colouring of AT_k^u with colours *brown, left, right, white* and *green.* The root of AT_k^u is coloured *brown*, leaves are coloured *green.* For every gate on an even layer, if it is the left (resp. right) child of its parent then colour it *left* (resp. *right*). Every gate on an odd layer, except the root, is coloured *white.* Denote this coloured alternately-binary-unary-tree as AT_k^c.

We define a circuit to be *properly* coloured if the root is coloured *brown*, leaves are coloured *green*, all multiplication gates but the root are coloured *white* and all addition gates are coloured *left* or *right* depending on whether they are left or right child respectively.

We obtain a *properly* coloured circuit from the universal circuit C_n as follows. For all addition gates in C_n we make two coloured copies, one coloured *left* and the other coloured *right.* We add edge connections as follows: for a multiplication gate we add an incoming edge to it from the *left* (resp. *right*) coloured copy of the left (resp. right) child, and for an addition gate the coloured gates are connected as the original gate in the circuit C_n.

We say that an undirected complete graph H on M nodes is *properly* coloured if, for all $s_n \leq M/2$, there is an embedding of the graph that underlies an s_n-sized *properly* coloured universal circuit, into H.

▶ **Theorem 15.** *Consider the homomorphism polynomial where*
- $G := \mathsf{AT}_m^c$.
- *H is a* properly *coloured complete graph (undirected) on poly(m), say m^6, nodes.*
- *$\mathcal{H}$:= set of coloured homomorphisms from G to H.*
- *α is 1 everywhere.*

Then, the family $(f_{\mathsf{AT}_m^c,H,\alpha,\textbf{\textit{ColHom}}}(\bar{X},\bar{Y}))_m$, where $m \in \mathbb{N}$, is VP*-hard for projections.*

Proof Sketch. As before, Y variables pick out C_m from H. The brown and green colours ensure that the root and the leaves of G are mapped to the root and leaves of C_m respectively. Injectivity follows from an argument similar to the one used for Theorem 14. ◀

The generic homomorphism polynomial gives us immense freedom in the choice of G, target graph H, weights α and the set of homomorphisms $\mathcal{H}$. Until now we used several modified graphs along with different restrictions on $\mathcal{H}$ to capture computations in the universal circuit. The question here is: can we get rid of restrictions on the set of homomorphisms? We provide a positive answer, using instead weights on the vertices of the source graph.

Homomorphism with weights

For k a power of 2, let T_k denote a complete (perfect) binary tree with k leaves.

▶ **Theorem 16.** *Consider the homomorphism polynomial where*

- $G := \mathsf{T}_m$.
- H *is a complete graph (undirected) on poly(m), say* m^6*, nodes.*
- $\mathcal{H}$:= *set of all homomorphisms from* G *to* H.
- *Define* α *such that,*

$$\alpha(u) = \begin{cases} 0 & u = \textit{root} \\ 1 & \textit{if } u \textit{ is the right child of it's parent} \\ 0 & \textit{otherwise} \end{cases}$$

Then, the family $(f_{\mathsf{T}_m,H,\alpha,\boldsymbol{Hom}}(\bar{X},\bar{Y}))_m$ *is* VP*-hard for constant-depth c-reductions.*

Proof Sketch. Since the source graph is complete binary trees, we first need to compact parse trees and get rid of the unary nodes (corresponding to + gates). We construct from the universal circuit C_n a graph J_n that allows us to get rid of the alternating binary-unary parse tree structure while maintaining the property that the compacted "parse trees" are subgraphs of J_n. The graph J_n has two copies g_L and g_R of each $\times$ gate and input gate of C_n. It also has two children attached to each leaf node. The edges of J_n essentially shortcut the + edges of C_n.

As before, we use Y variables to pick out J_n from H. We assign special variables w on edges from the root to a node g_R, and z on edges going from a non-root non-input node u to some right copy node g_R. For an input node g in the "left sub-graph" of J_n, the new left and right edges are assigned c_ℓ and x respectively, where x is the corresponding input label of g in C_n, and the node at the end of the x edge is assigned a special variable y. In the right sub-graph, variable c_r is used.

We show that homomorphisms whose monomials have degree 1 in w, 2^k-2 in z, 2^{k-1} each in c_ℓ and c_r, and 2^k in y are in bijection with compacted parse trees in J_n. The argument proceeds in stages: first show that the homomorphism is well-rooted (using the degree constraint on w, c_ℓ, c_r and the 0-1 weights in G), then show that it preserves layers (does not fold back) (using the degree constraint on c_ℓ, c_r and y), then show that it is injective within layers (using the degree constraint in z and the 0-1 weights in G). ◀

4 Upper Bounds: membership in VP

In this section we will show that most of the variants of the homomorphism polynomial considered in the previous section are also computable by polynomial size arithmetic circuits. That is, the homomorphism polynomials are VP-Complete. For sake of clarity we describe the membership of a generic homomorphism polynomial in VP in detail. Then we explain how to obtain various instantiations via projections.

We define a set of new variables $\bar{Z} := \{Z_{u,a} \mid u \in V(G) \text{ and } a \in V(H)\}$. Let us generalise the homomorphism polynomial $f_{G,H,\alpha,\mathcal{H}}$ as follows:

$$f_{G,H,\mathcal{H}}(\bar{Z},\bar{Y}) = \sum_{\phi\in\mathcal{H}} \left(\prod_{u\in V(G)} Z_{u,\phi(u)}\right)\left(\prod_{(u,v)\in E(G)} Y_{\phi(u),\phi(v)}\right).$$

Note that for a 0-1 valued α, we can easily obtain $f_{G,H,\alpha,\mathcal{H}}$ from our generic homomorphism polynomial $f_{G,H,\mathcal{H}}$ via substitution of $\bar{Z}$ variables, setting $Z_{u,a}$ to $X_a^{\alpha(u)}$. (If α can take any non-negative values, then we can still do the above substitution. We will need subcircuits computing appropriate powers of the $\bar{X}$ variables. The resulting circuit will still be poly-sized and hence in VP, provided the powers are not too large.)

▶ **Theorem 17.** *The family of homomorphism polynomials* $(f_m) = f_{G_m,H_m,\textbf{Hom}}(\bar{Z},\bar{Y})$ *where*

- G_m *is* T_m*, the complete balanced binary tree with* $m = 2^k$ *leaves,*
- H_m *is* K_n*, complete graph on* $n = poly(m)$ *nodes,*

is in VP*.*

Proof Sketch. The idea is to group the homomorphisms based on where they send the root of G_m and its children, and to recursively compute sub-polynomials within each group. The sub-polynomials in a specific group will have a specific set of variables in all their monomials. Thus the group can be identified by suitably combining partial derivatives of the recursively constructed sub-polynomials. (Note: this is why we consider the generalised polynomial with $\bar{Z}$ instead of $\bar{X}$ and α. If for some u, $\alpha(u) = 0$, then we cannot use partial derivatives to force sending u to a specific vertex of H.) The partial derivatives themselves can be computed efficiently using Lemma 3. ◀

▶ Remark. In the above theorem and proof, if G_m is AT^u_m instead of T_m, essentially the same construction works. The grouping of homomorphisms should be based on the images of the root and its children and grandchildren as well.

If G_m and H_m have directions, again everything goes through the same way.

If we want to consider a restricted set $\mathcal{H}$ of homomorphisms **DirHom** or **ColHom** instead of all of **Hom**, again the same construction works. All we need is that $\mathcal{H}$ can be decomposed into independent parts with a local stitching-together operator. That is, whether ϕ belongs to $\mathcal{H}$ can be verified locally edge-by-edge and/or vertex-by-vertex, so that this can be built into the decomposition and the recursive construction.

From Theorem 17, the discussion preceding it and the remark following it, we have:

▶ **Corollary 18.** *The polynomial families from Proposition 12, Theorems 14, 15, and 16 are all in* VP*.*

▶ Remark. It is not clear how to get a similar upper bound for **InjDirHom** when the target graph is the complete directed graph (remark following Proposition 12), or for the family from Proposition 13. We need a way of enforcing that the recursive construction above respects injectivity. This is not a problem for Proposition 12, though, because there the target graph is the graph underlying a multiplicatively disjoint circuit. Injectivity at the root and its children and grandchildren can be checked locally; the recursion beyond that does not fold back because the homomorphisms are direction-preserving. The construction may not work if the target graph is the complete directed graph.

From Corollary 18, Proposition 12, and Theorems 14, 15 and 16, we get our main result:

▶ **Theorem 19.**
1. *The polynomial families from Proposition 12 and Theorem 15 are complete for* VP *with respect to p-projections.*
2. *The polynomial families from Theorems 14 and 16 are complete for* VP *with respect to constant-depth c-reductions.*

5 Characterizing other complexity classes

We complement our result of VP-completeness by showing that appropriate modification of G can lead to VBP-complete and VNP-complete polynomial families.

VBP Completeness

VBP is the class of polynomials computed by polynomial-sized algebraic branching programs. These are layered directed graphs, with edges labeled by field constants or variables, and with a designated source node s and target node t. For any path ρ in G, the monomial $mon(\rho)$ is the product of the labels of all edges in ρ. For two nodes u, v, the polynomial p_{uv} sums $mon(\rho)$ for all paths ρ from u to v. The branching program computes the polynomial p_{st}.

A well-known polynomial family complete for VBP is the determinant of a generic matrix. A generic complete polynomial for VBP is the polynomial computed by an ABP with (1) a source node s, $m-1$ layers of m nodes each, and a target node t, (2) complete bipartite graphs between layers, and (3) distinct variables $\bar{x}$ on all edges. This is also the iterated matrix multiplication polynomial IMM. It is easy to see that st paths play the same role here as parse trees did in the multiplicatively disjoint circuits.

▶ **Theorem 20.** *Consider the homomorphism polynomial where*

- *G is a simple path on $m+1$ nodes, $(u_1, u_2, \ldots, u_{m+1})$.*
- *H is a complete graph (undirected) on m^2 nodes.*
- *$\mathcal{H}$:= set of all homomorphisms from G to H.*
- *Define α such that $\alpha(u) = \begin{cases} 1 & u = u_1 \text{ or } u = u_{m+1} \\ 0 & \text{otherwise} \end{cases}$*

Then, the family $(f_{G,H,\alpha,\mathbf{Hom}}(\bar{X}, \bar{Y}))_m$, where $m \in \mathbb{N}$, is complete for VBP *under c-reductions.*

Proof Sketch. The hardness proof is very similar to that in Theorem 16. Lemma 3 (and hence Theorem 17) doesn't work for branching programs; however we show membership by direct construction of an ABP. ◀

VNP Completeness

▶ **Theorem 21.** *Consider the homomorphism polynomial where*

- *G is the complete graph (undirected) on m nodes.*
- *H is the complete graph (undirected) on m nodes.*
- *$\mathcal{H}$:= set of all homomorphisms from G to H.*
- *All $\bar{Y}$ variables are set to 1.*

Then, the family $(f_{G,H,\mathbf{Hom}}(\bar{Z}))_m$, where $m \in \mathbb{N}$, is complete for VNP *under p-projections.*

Proof Sketch. This homomorphism polynomial is exactly the per_m polynomial. ◀

6 Conclusion

We have shown that several natural homomorphism polynomials are complete for the algebraic complexity class VP. Our results are summarised below.

Complexity	G	H	$\mathcal{H}$	polynomial type	reduction
VP-complete	AT_m	$C_{m^{O(1)}}$	**InjDirHom**	$\alpha = \underline{1}$	p-projections
	AT^d_m	$DK_{m^{O(1)}}$	**DirHom**	$\alpha = \underline{1}$	$O(1)$-depth c-reductions
	AT^c_m	coloured $K_{m^{O(1)}}$	**ColHom**	$\alpha = \underline{1}$	projections
	T^u_m	$K_{m^{O(1)}}$	**Hom**	0-1 valued	$O(1)$-depth
VBP-complete	Path_m	$K_{m^{O(1)}}$	**Hom**	0-1 valued	$O(1)$-depth
VNP-complete	K_m	K_m	**Hom**	generalised ($\bar{Z}$ variables)	p-projections

It would be interesting to show all the hardness results with respect to p-projections. It would also be very interesting to obtain completeness while allowing all homomorphisms on simple graphs and eliminating vertex weights. Another question is extending the completeness results of this paper to fields of characteristic other than zero.

Perhaps more importantly, it would be nice to get still more examples of natural VP-complete problems, preferably from different areas. The completeness of determinant or iterated matrix multiplication for VBP underlies the importance of linear algebra as a source of "efficient" computations. Finding natural VP-complete polynomials in some sense means finding computational techniques which are (believed to be) stronger than linear algebra.

References

1 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983.

2 Christian Borgs, Jennifer T. Chayes, László Lovász, Vera T. Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in Discrete Math*, pages 315–371. Springer, 2006.

3 P. Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*, volume 7 of *Algorithms and Computation in Mathematics*. Springer, 2000.

4 Florent Capelli, Arnaud Durand, and Stefan Mengel. The arithmetic complexity of tensor contractions. In *Symposium on Theoretical Aspects of Computer Science STACS*, volume 20 of *LIPIcs*, pages 365–376, 2013.

5 Nicolas de Rugy-Altherre. A dichotomy theorem for homomorphism polynomials. In *Mathematical Foundations of Computer Science 2012*, volume 7464 of *LNCS*, pages 308–322. Springer Berlin Heidelberg, 2012.

6 Arnaud Durand and Stefan Mengel. The complexity of weighted counting for acyclic conjunctive queries. *J. Comput. Syst. Sci.*, 80(1):277–296, 2014.

7 Martin E. Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM J. Comput.*, 42(3):1245–1274, 2013.

8 Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. *Journal of Complexity*, 24(1):16–38, 2008.

9 Stefan Mengel. Characterizing arithmetic circuit classes by constraint satisfaction problems. In *Automata, Languages and Programming*, volume 6755 of *LNCS*, pages 700–711. Springer Berlin Heidelberg, 2011.

10 Ran Raz. Elusive functions and lower bounds for arithmetic circuits. *Theory of Computing*, 6:135–177, 2010.

11 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

12 Leslie G. Valiant. Completeness classes in algebra. In *Symposium on Theory of Computing STOC*, pages 249–261, 1979.

13 Leslie G. Valiant. Reducibility by algebraic projections. In *Logic and Algorithmic: International Symposium in honour of Ernst Specker*, volume 30, pages 365–380. Monograph. de l'Enseign. Math., 1982.

14 Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, 1983.

Computing Information Flow Using Symbolic Model-Checking

Rohit Chadha[1], Umang Mathur[2], and Stefan Schwoon[3]

1 **University of Missouri, USA**
2 **Indian Institute of Technology - Bombay, India**
3 **LSV, ENS Cachan & CNRS, INRIA Saclay, France**

Abstract

Several measures have been proposed in literature for quantifying the information leaked by the public outputs of a program with secret inputs. We consider the problem of *computing* information leaked by a deterministic or probabilistic program when the measure of information is based on (a) min-entropy and (b) Shannon entropy. The key challenge in computing these measures is that we need the total number of possible outputs and, for each possible output, the number of inputs that lead to it. A direct computation of these quantities is infeasible because of the state-explosion problem. We therefore propose symbolic algorithms based on binary decision diagrams (BDDs). The advantage of our approach is that these symbolic algorithms can be easily implemented in any BDD-based model-checking tool that checks for reachability in deterministic non-recursive programs by computing program summaries. We demonstrate the validity of our approach by implementing these algorithms in a tool `Moped-QLeak`, which is built upon `Moped`, a model checker for Boolean programs. Finally, we show how this symbolic approach extends to probabilistic programs.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Information leakage, Min Entropy, Shannon Entropy, Abstract decision diagrams, Program summaries

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.505

1 Introduction

It is desirable for a program to never leak any information about its confidential inputs. For example, when an adversary can make *low-security observations* of an execution, these should be independent of the confidential inputs. This property is often too strong in practice, mostly because it clashes with desired functionality. Therefore, many authors (cf. [22, 25]) have proposed to evaluate security by the *amount* of leaked confidential information. This raises foundational questions of (a) how to measure that amount and (b) how to compute it. These challenges have received much attention recently.

A usual approach is to employ information-theoretic tools. In this approach, a program is modeled as an information channel that transforms a random variable taking values from the set of confidential inputs into a random variable taking values from the set of public outputs (i. e., the adversary's observations). Based on this, one quantifies the adversary's uncertainty about the confidential inputs. The amount of information leaked by the program is then modeled as the difference between the initial uncertainty and the uncertainty remaining in the secret inputs after the adversary observes the execution. Commonly used measures of uncertainty are Shannon entropy [22] and min-entropy [25]. Intuitively, leakage based on min-entropy measures vulnerability of the secret inputs to a single guess of the adversary who observes the program execution, while, leakage based on Shannon entropy measures expected number of guesses required for the adversary to guess the secret input having observed the program execution. We refer to [25] for a detailed comparison between these two measures.

Though appealing from a conceptional viewpoint, these measures do not readily lend themselves to feasible computation. For example, it has been shown in [27] that when using Shannon's entropy for measuring uncertainty, the problem of deciding whether the information leaked by a *loop-free* deterministic Boolean program is less than a rational number is harder than *counting* the number of satisfying assignments of a Boolean formula in Conjunctive Normal Form. The hardness of the problem comes from the fact that one has to compute (a) how many outputs are observable to the adversary and (b) for each possible output, how many inputs lead to that particular output.

When Boolean deterministic programs contain loops, computing information leakage becomes PSPACE-complete [28, 6, 26], for both min-entropy and Shannon entropy. Although this is same complexity as checking safety of Boolean programs (or equivalently, reachability), the decision procedures given in [28, 6, 26] are not feasible in practice. Instead researchers have developed heuristics to exploit reachability tools to compute the amount of information leaked. The reachability tools employed come from model checking [3, 18, 8, 9], static analysis [11, 3], SMT solvers [21, 20, 23, 16], and statistical analysis [18, 7].

Contributions. We first consider the problem of evaluating the amount of information leaked by the public outputs of Boolean *deterministic* programs with uniformly distributed secret inputs. We exploit symbolic model-checking techniques to achieve our goals. More precisely, we demonstrate how model checkers based on Binary Decision Diagrams (BDDs) can very easily be enhanced to compute information leakage. As we shall see shortly, our approach is informed by the model-checking algorithms used by these tools.

BDDs [19, 1, 5] are data structures used to store Boolean functions. Their efficiency has led to many applications in program verification. Broadly, in this approach, the program is viewed as a transition system in which a configuration contains the current line number and the values of the variables. Transitions are encoded as BDDs, and reachability is encoded as the least fixed-point solution to a set of Boolean equations. This solution is the result of a fixed-point iteration with efficient BDD operations (Please see [5] for a discussion of complexity of BDD operations). For certain BDD-based tools (cf. [13]), this fixed-point computation yields the relation between the values of global variables at the start of the program and the values of the global variables when the queried location is reached. By querying the exit point, we can thus compute the relation between the inputs and the outputs of the program, henceforth referred to as the *summary* of the program.

Our key observation is that this summary (which is given as a BDD) is indeed all the information we need to quantify information leakage. We give symbolic algorithms that extract information leakage from the summary according to either Shannon entropy or min-entropy. This approach is appealing because these algorithms can be easily plugged into existing BDD-based model-checking tools.

We validate this approach by implementing our algorithms in `Moped` [13], a BDD-based symbolic model checker that checks for assertion errors in programs modeled as reachability problems. Apart from providing support for Boolean data, `Moped` also supports integers of variable length, arrays, and C-like structures. Our experience with these implementations are promising, as the computation of information leakage (for both min-entropy and Shannon-entropy) comes with little overhead over the reachability computation.

We then turn our attention to probabilistic non-recursive programs. For such programs, we need to compute, for each possible input-output pair (i, o), the conditional probability that the program outputs o when the input is i. Usually, these quantities are stored as a matrix, also called the *channel matrix*. We compute the channel matrix as the least fixed-point solution to a system of *linear equations* [24], which can be done using Algebraic Decision Diagrams (ADDs) [12], a generalization of BDDs. The summary for probabilistic programs now encodes (symbolically) the channel matrix, and we construct symbolic algorithms to extract the leaked information from the computed summary. We validate this approach by extending the ability of `Moped` to compute the summary for probabilistic non-recursive programs and implementing the symbolic algorithms for computing the information leakage.

The tool implementing the algorithms for entropy calculations is available for download at `http://people.cs.missouri.edu/~chadhar/mql/`. For space reasons, we have omitted proofs which can be found in the longer version of the paper available at the same site.

Related work. The problem of automatically computing information flow was first tackled in [3]. This approach iteratively constructs equivalence classes on inputs: two inputs are said to be equivalent if they lead to the same output. One starts with a single equivalence class and progressively refines when these inputs lead to different outputs. At each step, the equivalence relation is characterized using logical formulas and refined using experimental runs of the program. Once a fixed point is reached, the sizes of the resulting equivalence classes can be used to compute information leakage. This technique is optimized in [18], where statistical techniques are used to estimate the equivalence classes. The effectiveness of the approach is demonstrated through examples, and the authors suggest that an automated tool based on these techniques can be built. The computation of the size of the equivalence classes is further optimized in [15].

SMT solvers are used in [21, 20, 16, 23] to estimate min-entropy leakage in Boolean *straight-line* programs. In this approach, the program summary is encoded as a SMT formula and various model-counting techniques are used to obtain the information leaked. In [21, 20], an upper bound on min-entropy leakage is computed by estimating an upper bound on the number of feasible outputs. It is easy to construct examples where the computed bound is far from the correct value. Our techniques in contrast yield exact values. [16] provides a toolchain which first computes the program summary as a SAT formula that is then fed to a custom-made #SAT solver to calculate the information leaked. [23] combines model-counting techniques of #SMT solvers with the technique of symbolic executions. This tool can handle real C and Java programs.

For probabilistic programs, the use of model-checking to compute information leakage has been explored in [8, 2, 10, 4]. Please note that the models considered in these papers are more general as they also allow for other observations than just the outputs at the end of the program execution. [8] uses [14] to get the channel matrix and then computes the information leakage by hand, [10] implements an explicit state model-checking algorithm, and [4] computes the information leakage using (forward) symbolic executions. [2] also proposes to compute the channel matrix using fix-point iterations. Once the channel matrix is computed *explicitly* then information leakage can be computed. Our approach is different in that we solve the fix-point iterations *symbolically* and use the *symbolic* representation of the computed matrix directly in the computation of information leakage.

2 Preliminaries

We recall some standard definitions and establish notations. For a finite set A, $|A|$ shall denote the number of elements of A. For a function $f : A \to B$ and $b \in B$, $f^{-1}(b)$ denotes the set $\{a \mid f(a) = b\}$. We write 2^A to mean both the set of functions $A \to \{\texttt{true}, \texttt{false}\}$ as well as the set of subsets of A. All logarithms are to the base 2 and $0 \log 0 := 0$. The set of real numbers will be denoted by $\mathbb{R}$ and the non-negative reals by $\mathbb{R}^+$.

Boolean Programs. We first consider non-recursive Boolean deterministic programs in this paper. The programs that we consider have global and local variables. Usually, it is assumed that the set of global variables is partitioned into two: set of *high-security input-only variables* and the set of *low-security output-only variables*. However, in this paper we will assume that the secret inputs to the program are the *initial* values of the global variables and that the public outputs are the *final* values of the global variables. Thus, we do not explicitly separate high-security input-only variables and low-security output-only variables. This does not cause a loss of expressiveness: if we want to make sure that changes to a high-security input-only variable by a program are not visible to the adversary, we can set them to `false` upon exit. Similarly, if we want to explicitly designate some variables as low-security output-only variables, we can initialize all of them to `false`.

Assuming that local variables are always initialized to `false`, the semantics of a program P with global variables $\mathcal{G}$ can be seen as a function $F_P\colon S \to O$ where $S = 2^{\mathcal{G}}$ and $O = 2^{\mathcal{G}} \cup \{\bot\}$. $F_P(\bar{g_0}) = \bot$ iff P does not terminate on $\bar{g_o}$, otherwise $F_P(\bar{g_0}) \in 2^{\mathcal{G}}$ is the valuation of the global variables when P stops executing. From now on, we will confuse P with the function F_P. Note that we treat non-termination as an explicit observation of the attacker.

Assuming that the inputs are sampled from a distribution μ, let $\mathcal{S}$ be the random variable taking values in S according to μ. μ can be extended to a joint probability distribution on S and O by setting $\mu(\mathcal{O} = o \mid \mathcal{S} = s) = 1$ if $P(s) = o$ and 0 otherwise.

Information leakage in programs. Several measures of information leakage have been considered in literature. Of these, we consider Shannon entropy and min-entropy. We assume that the reader is familiar with information theory and introduce some abbreviations and results that we shall need. For this section, we fix a Boolean program P. As discussed above, the semantics of P is a function $P : S \to O$. If S is sampled from a distribution μ, then μ gives rise to a joint probability distribution on S and O.

Leakage based on Shannon entropy: In Shannon entropy, the *information leaked by the program P* is defined as $\mathrm{SE}_\mu(P) := \mathrm{I}_\mu(\mathcal{S}; \mathcal{O})$, where $\mathcal{S}$ and $\mathcal{O}$ are random variables taking values in S and O respectively according to the joint distribution μ, and $\mathrm{I}_\mu(\mathcal{S}; \mathcal{O})$ is the mutual information of random variables $\mathcal{S}$ and $\mathcal{O}$. When P is deterministic and μ is U, the uniform distribution on inputs, we have [3, 17]: $\mathrm{SE}_{\mathsf{U}}(P) = \log |S| - \frac{1}{|S|} \sum_{o \in O} |P^{-1}(o)| \log |P^{-1}(o)|$.

Leakage based on Min entropy: In min-entropy [25], the *information leaked by the program P* on uniformly distributed inputs is defined as $\mathrm{ME}_{\mathsf{U}}(P) := \log \sum_{o \in \mathcal{O}} \max_{s \in \mathcal{S}} \mu(\mathcal{S} = s \mid \mathcal{O} = o)$. When P is deterministic [25], we get that $\mathrm{ME}_\mu(P) := \log |\mathcal{O}'|$, where $\mathcal{O}' = \{o \in \mathcal{O} \mid \exists s \in \mathcal{S} : P(s) = o\}$ are the outputs that can actually be realized.

Algebraic Decision Diagrams. We assume that the reader is familiar with Binary Decision Diagrams (BDDs) and merely recall some facts necessary for our presentation. Our presentation follows closely the presentation in [24]. When speaking of BDDs, we always mean their reduced ordered form [5]. BDDs are data structures for storing elements of $2^{\mathcal{V}} \to \{0, 1\}$, where $\mathcal{V} = \{x_1, \ldots, x_n\}$ is a finite set of Boolean variables. They take the form of a rooted, directed acyclic labeled graph. Non-terminal nodes are labeled by an element of $\mathcal{V}$, and terminals are either 0 or 1. There are two edges out of a non-terminal node, one labeled *then* and the other labeled *else*. Assuming a fixed strict order $<$ on $\mathcal{V}$, an edge from a non-terminal

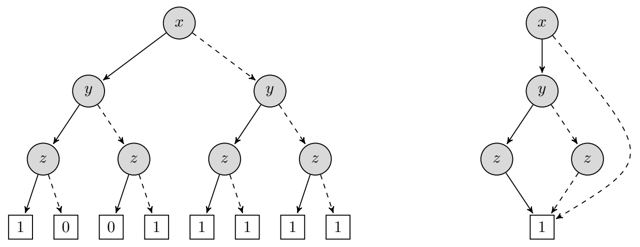


Figure 1 An unreduced decision diagram (left) and a corresponding BDD (right).

labeled x to a non-terminal labeled y satisfies $x < y$. From now on we will often confuse a function $2^{\mathcal{V}} \to \{0,1\}$ with its BDD representation.

▶ **Example 1.** Figure 1 shows how a BDD over the set $\mathcal{V} = \{x, y, z\}$ with the order $x < y < z$ would store the Boolean assignments satisfying $x \to (y \leftrightarrow z)$. The figure on the left shows a (non-reduced) diagram exhaustively listing all assignments, and the right-hand side shows the resulting BDD, where for simplicity the terminal 0 and edges leading to it have been omitted. The solid arrows are *then* branches and the dashed arrows are *else* branches.

ADDs generalize BDDs and store elements of the set $2^{\mathcal{V}} \to M$, where $\mathcal{V} = \{x_1, \ldots, x_n\}$ and M is an arbitrary set. The main difference between BDDs and ADDs is that the terminal nodes contain elements of M and not just elements of $\{0,1\}$. For our purposes, M will be either $\mathbb{R}$ or $\mathbb{R}^+$. Analogous to BDDs, the value of a function f represented by an ADD T at $(z_1, \ldots, z_n) \in 2^{\mathcal{V}}$ is given by the label of the terminal node along the unique path from the root to a terminal node such that if a non-terminal node is labeled x_i along the path then the outgoing edge from x_i must be labeled *then* if and only if z_i is `true`.

Note that an BDD is an ADD where all the terminals are either 0 or 1. Henceforth, we will refer to BDDs as 0/1-ADDs. Many efficient operations can be performed on ADDs. We list the most relevant ones for our paper.

1. The function $\mathsf{isConst}(T)$ checks if T is a constant function. $\mathsf{val}(T)$ returns the value of T if $\mathsf{isConst}(T)$ is true.
2. If op is a commutative and associative binary operator on $\mathbb{R}$ and $\mathcal{V}_1$ a subset of variables of $\mathcal{V}$ then $\mathsf{abstract}(op, \mathcal{V}_1, T)$ returns the result of abstracting all the variables in $\mathcal{V}_1$ by applying the operator op over all possible values taken by variables in $\mathcal{V}_1$. $\mathsf{abstract}(op, \mathcal{V}_1, T)$, thus obtained, is a function with domain as the set $\mathcal{V} \setminus \mathcal{V}_1$ and range as $\mathbb{R}$.
   For example, if T represents the function f, then $\mathsf{abstract}(+, \{x_1, x_2\}, T)$ returns the ADD which represents the function $f(\texttt{true}, \texttt{true}, x_3, \ldots, x_n) + f(\texttt{true}, \texttt{false}, x_3, \ldots, x_n) + f(\texttt{false}, \texttt{true}, x_3, \ldots, x_n) + f(\texttt{false}, \texttt{false}, x_3, \ldots, x_n)$.
3. If T is a 0/1-ADD and $\mathcal{V}_1$ a subset of $\mathcal{V}$, then $\mathsf{orAbstract}(\mathcal{V}_1, T)$ returns the result of abstracting all the variables in $\mathcal{V}_1$ by applying disjunction over all possible values taken by variables in $\mathcal{V}_1$.

3 Leakage in non-probabilistic programs

In this section, we shall describe our ADD-based algorithms for computing the information leaked by deterministic programs when the leakage is measured using (a) min-entropy and (b) Shannon entropy. We fix some notation. Consider a set of variables $\mathcal{G} = \{x_1, \ldots, x_n\}$. Let $\mathcal{G}' = \{x'_1, \ldots, x'_n\}$ be a set of distinct variables disjoint from $\mathcal{G}$. Note that there is a one-to-one correspondence between elements of $2^{\mathcal{G}}$ and $2^{\mathcal{G}'}$ and every element $(z'_1, \ldots, z'_n)$

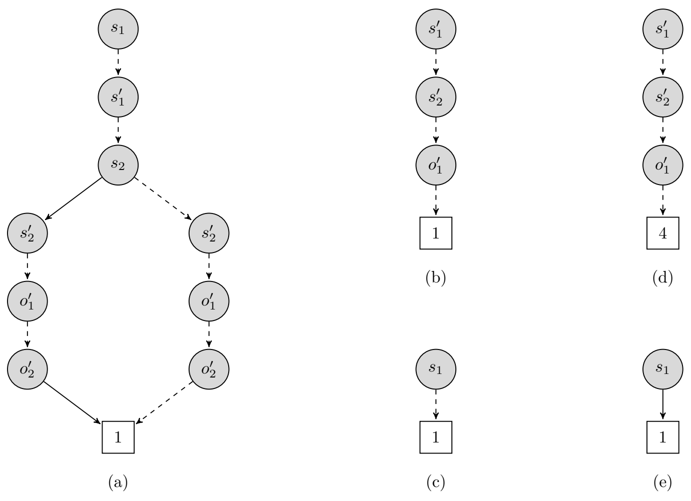


Figure 2 (a) Transition relation of the program P_{ex}. The ordering assumed is $s_1 < s'_1 < s_2 < s'_2 < o_1 < o'_1 < o_2 < o'_2$. (b) All the possible outputs of the program P_{ex} as an ADD. (c) All possible inputs on which P_{ex} terminates represented as an ADD. (d) $T_{\textsf{eq-size},P}$ for P_{ex} as an ADD. (e) $T_{\textsf{non-term},P}$ for the program P_{ex} as an ADD.

of $2^{\mathcal{G}'}$ can be identified with a unique element $(z_1, \ldots, z_n)$ of $2^{\mathcal{G}}$ and vice versa. $\mathcal{G}$ shall represent the initial values of the variables of a program and $\mathcal{G}'$ shall represent their final values. In this section, we will assume that all possible valuations of $\mathcal{G}$ are valid inputs to the program (and hence our input domain shall always be a power of 2). We discuss how to restrict the domain in the longer version of the paper.

▶ **Definition 2** (Summary of a Program)**.** Let P be a program with $\mathcal{G} = \{x_1, \ldots, x_n\}$ as the set of global variables. Let $\mathcal{G}' = \{x'_1, \ldots, x'_n\}$ be a set of distinct variables disjoint from $\mathcal{G}$. The *summary* of P, denoted T_P, is a function $T_P : 2^{(\mathcal{G} \cup \mathcal{G}')} \to \{0, 1\}$ such that for every $z_1, \ldots, z_n, z'_1, \ldots, z'_n \in \{\texttt{true}, \texttt{false}\}$, we have $T_P(z_1, \ldots, z_n, z'_1, \ldots, z'_n) = 1 \iff P(z_1, \ldots, z_n) = (z'_1, \ldots, z'_n)$.

Observe that thanks to the correspondence between OBDDs and Boolean functions, T_P can be considered as an OBDD on the set of variables $\mathcal{G} \cup \mathcal{G}'$. Now, T_P can be seen as the least fixed point of a system of Boolean equations, which can efficiently be constructed by iterative methods. For our purposes, it suffices to say that BDD-based model-checkers essentially construct this relation for us (and, if not, can be modified to carry out this construction). We assume for our paper that T_P is constructed by a BDD-based model-checker. It remains to show how to exploit T_P to compute the information leaked by P.

▶ **Example 3.** Consider the following Boolean program P_{ex} with global variables: $s1, s2, o1$ and $o2$.

```
o1 = false; o2 = false;
while s1 {};
o1 = false; o2 = s2;
s1 = false; s2 = false;
```

Here, variables $s1$ and $s2$ are high-security input-only variables and $o1$, $o2$ low-security input variables. This is why we initialized o_1 and o_2 to be `false` and set s_1 and s_2 `false` before the end of the program. Observe also that the program does not terminate when $s1$ is `true` at the beginning of the program. Assuming the order $s_1 < s'_1 < s_2 < s'_2 < o_1 < o'_1 < o_2 < o'_2$, the transition relation of P is shown as a 0/1-ADD in Figure 2 (a).

For the rest of the section, unless otherwise stated, we will fix the Boolean program P. We assume that $\mathcal{G} = \{x_1, \dots, x_n\}$ is the set of global variables of P and that $\mathcal{G}' = \{x'_1, \dots, x'_n\}$ is a set of distinct variables disjoint from $\mathcal{G}$. The *summary* of P will be referred to as T_P.

Leakage measured using min-entropy. The amount of information leaked by the program P when using min-entropy as measure of information is as follows. Let $\mathrm{post}(2^{\mathcal{G}}) = \{\bar{g}' \in 2^{\mathcal{G}} \mid \exists \bar{g} \in 2^{\mathcal{G}}.\ P(\bar{g}) = \bar{g}'\}$. If the program P terminates on all inputs then the min-entropy leakage is $\log|\mathrm{post}(2^{\mathcal{G}})|$, otherwise it is $\log(|\mathrm{post}(2^{\mathcal{G}})| + 1)$.

Thus, to compute the min-entropy leakage, we need to compute $|\mathrm{post}(2^{\mathcal{G}})|$ and check if there is an input on which the program P never terminates. The following lemma shows how these two tasks can be achieved using ADDs.

▶ **Lemma 4.** *Let $T_{out,P}$ = orAbstract($\mathcal{G}, T_P$) and $T_{term,P}$ = orAbstract($\mathcal{G}', T_P$).*
1. $|\mathrm{post}(2^{\mathcal{G}})|$ = *val(abstract($+, \mathcal{G}', T_{out,P}$)).*
2. *P terminates on every input iff isConst($T_{term,P}$) and val($T_{term,P}$) $= 1$.*

▶ **Example 5.** Consider the program P_{ex} in Example 3 with $\mathcal{G} = \{s_1, s_2, o_1, o_2\}$. Observe that the program terminates only when s_1 is `false`, in which case the final value of s_1 is also `false`. The initial values of o_1 and o_2 do not effect the output. The final values of s_2 and o_1 are always `false`. The value of o_2 is exactly the value of s_2. Thus, there are two possible outputs (`false`, `false`, `false`, `true`) and (`false`, `false`, `false`, `false`), both of which happen for exactly 4 inputs. The ADD representing $T_{\mathsf{out},P}$, the set of all possible outputs of P is given in Figure 2 (b). Note that o'_2 does not appear in the picture because the *then* and *else* branches of o'_2 lead to isomorphic subtrees. Observe that abstract($+, \mathcal{G}', T_{\mathsf{out},P}$) is the constant ADD 2. The ADD $T_{\mathsf{term},P}$ representing all possible inputs on which P terminates is given in Figure 2 (c).

▶ **Theorem 6.** *For a program P with global variables $\mathcal{G} = \{x_1, \dots, x_n\}$, let $\mathcal{G}' = \{x'_1, \dots, x'_n\}$ be a set of distinct variables disjoint from $\mathcal{G}$. Let T_P be the summary of P represented as a 0/1-ADD on $\mathcal{G} \cup \mathcal{G}'$. The Algorithm 1 computes* $\mathrm{ME}_{\mathsf{U}}(P)$.

Algorithm 1: Symbolic computation of min-entropy leakage of a deterministic program

```
Input: $\mathcal{G}, \mathcal{G}'$ and $T_P$ the summary of $P$.
Output: $\mathrm{ME}_{\mathsf{U}}(P)$
1 begin
2     $T_{\mathsf{out},P} \longleftarrow$ orAbstract($\mathcal{G}, T_P$)
3     num_out $\longleftarrow$ val(abstract($+, \mathcal{G}', T_{\mathsf{out},P}$))
4     $T_{\mathsf{term},P} \longleftarrow$ orAbstract($\mathcal{G}', T_P$)
5     if isConst($T_{term,P}$) = false or val($T_{term,P}$) = 0 then
6         num_out $\longleftarrow$ num_out + 1;
7     return log num_out
```

Leakage measured using Shannon entropy. We now consider information leaked by P when measured using Shannon entropy. We need to compute $\sum_{\bar{g}' \in 2^{\mathcal{G}'}} |P^{-1}(\bar{g}')| \log|P^{-1}(\bar{g}')|$ $+|P^{-1}(\bot)| \log|P^{-1}(\bot)|$. In order to compute this sum, we need a new auxiliary definition:

▶ **Definition 7.** Let $\star : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}$ be the binary operator defined as $r_1 \star r_2 = r_1 \log r_1 + r_2 \log r_2$.

Algorithm 2: Symbolic computation of Shannon entropy leakage of a deterministic program

```
Input: $\mathcal{G}, \mathcal{G}'$ and $T_P$ the summary of $P$.
Output: $\mathrm{SE}_{\mathsf{U}}(P)$
1 Let $n$ be the number of variables in $\mathcal{G}$.
2 begin
3     $T_{\textsf{eq-size},P} \longleftarrow \textsf{abstract}(+, \mathcal{G}, T_P)$
4     $\textsf{sum} \longleftarrow \textsf{val}(\textsf{abstract}(\star, \mathcal{G}', T_{\textsf{eq-size},P}))$
5     $T_{\textsf{term},P} \longleftarrow \textsf{orAbstract}(\mathcal{G}', T_P)$
6     $T_{\textsf{non-term},P} \longleftarrow \textsf{cmpl}(T_{\textsf{term},P})$
7     $\textsf{num}_{\textsf{non-term}} \longleftarrow \textsf{val}(\textsf{abstract}(+, \mathcal{G}, T_{\textsf{non-term},P}))$
8     $\textsf{sum} \longleftarrow \textsf{sum} + \textsf{num}_{\textsf{non-term}} \log(\textsf{num}_{\textsf{non-term}})$
9     return $(n - \frac{\textsf{sum}}{2^n})$
```

▶ **Theorem 8.** *For a program P with global variables $\mathcal{G} = \{x_1, \ldots, x_n\}$, let $\mathcal{G}' = \{x'_1, \ldots, x'_n\}$ be a set of distinct variables disjoint from $\mathcal{G}$. Let T_P be the summary of P represented as a 0/1-ADD on $\mathcal{G} \cup \mathcal{G}'$. Algorithm 2 computes* $\mathrm{SE}_{\mathsf{U}}(P)$.

▶ **Example 9.** Consider the program P_{ex} in Example 3. Recall that there are two possible outputs (`false`, `false`, `false`, `true`) and (`false`, `false`, `false`, `false`), both of which happen for exactly 4 inputs. The ADD $T_{\textsf{eq-size},P}$ for the P_{ex} is depicted in Figure 2 (d). The program does not terminate whenever s_1 is `false`. The ADD $T_{\textsf{non-term},P}$ is depicted in Figure 2 (e).

4 Experimental evaluation

In this section, we present some results based on our experiments for calculating the different leakage values using the tool **`Moped-QLeak`**. It is based on the existing BDD-based symbolic model-checker **`Moped`** [13]. **`Moped`**, apart from providing support for basic Boolean data, also supports complex data types such as integers of variable length, arrays and C-like structures. **`Moped`** uses the CUDD (Colorado University Decision Diagram) package to implement BDDs.

`Moped-QLeak` performs basic reachability analysis and generates a summary of an input program written in *Remopla*. This summary is then used to calculate the information leakage. Currently, **`Moped-QLeak`** only supports non-recursive programs and is currently available for download at: `http://people.cs.missouri.edu/~chadhar/mql/`

`Moped` translates Remopla programs into BDDs. **`Moped-QLeak`** re-uses this as a frontend, but internally works with the more generic ADDs to carry out the calculations. Other than that, we made the following optimizations with respect to the standard behavior of **`Moped`**.

Algebraic operations: The input language of **`Moped`** understands expressions using algebraic and Boolean operations. However, **`Moped`** was not conceived with large integer operands in mind, and we detected some inefficiencies in these translations for integer operands having large number of bits. These often drastically affected the overall time taken for calculation of the summary, in which cases we improved the translation. Also, for the purpose of experimental evaluation of the efficiency of **`Moped-QLeak`**, we encode all the examples with variables having large range as Boolean programs and note a striking change in the running times. Furthermore, **`Moped`** does not support integers with bit length > 30. Hence, all examples with bit length > 30 were also coded as Boolean programs.

Size of ADDs and variable orderings: As usual with symbolic methods, their efficiency is highly sensitive to the size of the decision diagrams generated during the course of the reachability analysis, which, in turn, may depend on the variable ordering. (Finding the most efficient ordering is a NP-hard problem). **`Moped`** does not automatically determine the

Table 1 Examples used for evaluation.

Example	Order	ME	SE	Time	Data types
Illustrative Example	I	3	2.03966e-05	0.215	bool
Electronic Purse	D	2	2	0.009	5 bit integers (Restricted)
Mix and Duplicate	S	16	16	0.041	bool
Binary Search	I	16	16	9.307	bool
Sanity Check	I	4	1.16797e-7	0.060	bool
Implicit Flow	D	2.80735	1.757e-07	0.016	30 bit integers
Implicit Flow	D	2.80735	4.67189e-08	0.190	bool
Masked Copy	I	16	16	0.038	bool
Sum Query	D	4.80735	4.35132	0.034	5 bit integers (Restricted)
Ten Random Outputs	D	3.32193	2.6355e-07	0.055	30 bit integers
Population Count	I/D			out-of-memory	bool

best variable ordering and gives the user the flexibility to choose the ordering. Hence, the examples for which the default ordering of variables (which entails the order of declaration of the variables in the source file) was the overhead, have been re-written with supposedly efficient variable orderings. The principal obstacle here is the computation of summary. The computation of leakage itself adds little overhead.

We illustrate our orderings using the variables O (for public outputs) and S (for private inputs). Let O_N (O_1) be the most (least) significant bit of O and likewise S_N (S_1) the most (least) significant bit of S. We primarily used two kinds of orderings:

- Contiguous ordering: This is the default ordering of the tool, where we set $O_1 < O'_1 < O_2 \cdots O'_N < S_1 < S'_1 < S_2 < \cdots < S'_N$.
- Interleaved ordering: In this ordering, we set $O_1 < O'_1 < S_1 < S'_1 < O_2 < O'_2 < S_2 < S'_2 < \cdots O_N < O'_N < S_N < S'_N$.

The choice of an ordering depends largely on the structure and semantics of the program. The ADDs produced are generally smaller if a variable v_1 is closer to a variable v_2 such that the value of v_1 depends on the value of v_2. Essentially, as long as variables are compared and assigned to constants in the program, the default ordering works very well and in that case we do not even attempt the interleaved ordering. For other examples, typically, we switch to interleaved ordering as contiguous ordering becomes inefficient very fast with the number of bits as the ADDs become very large. Going by this, we have also reordered variable declarations in an example (see Mix and Duplicate below) so that variables with a constant difference in the *indices* are closer.

Table 1 presents some selected benchmark programs that we used to test `Moped-QLeak`. The examples have been derived from [21]. The experiments were conducted on a 64-bit Xeon-X5650 2.67GHz Linux machine. Unless otherwise stated, S and O are 32-bit unsigned integers in all the programs. For each example, we give the name, the ordering, the Shannon entropy (SE) and min-entropy (ME) leakage values, the execution time of the tool in seconds, and the data types that occur in the example, which are either all Boolean or integers with a specified number of bits. If the example uses restricted domains then we mention it in the data types. The order is either the contiguous default order (D), the interleaved order (I), or another example-specific order (S). There is one example from [21], Population Count, for which the computation of summary never succeeds as there is no good variable ordering for that example. Note that we run the tool to compute the two leakage values separately and report the worse case. The time difference between the computation of the two values is almost always within a 3-4 microseconds.

Mix and Duplicate. The following program copies the XOR of the i^{th} and the $(i+16)^{th}$ bit of S to both the i^{th} and the $(i+16)^{th}$ bit of O.

```
O = ((S >> 16) ^ S) & 0xffff;
O = O | O << 16;
```

It is thus that the i^{th} and the $(i+16)^{th}$ bits of S and O are closely related. In fact, the ADDs formed after reordering the variables to $O_{17} < O'_{17} < O_1 < O'_1 < S_{17} < S'_{17} < S_1 \cdots O'_{16} < S_{32} < S'_{32} < S_{16} < S'_{16}$ have drastic reduction in the number of distinct nodes. Note that intuitively, half the input bits are leaked in the example (namely the XOR of i^{th} and $(i+16)^{th}$ bits of S). This intuition is confirmed by the results.

Binary Search. The following program scans the first b bits of the input S and puts a 1 at the i^{th} bit of O iff the i^{th} bit of S is 1.

```
O = 0;
for (i = 0; i < b; i++)
  {m = 2^(31-i);
  if (O + m <= S) O += m; }
```

For our experiments, we took $b = 16$. We converted this program to a Boolean program with an interleaved ordering. We also unrolled the loop for $b = 16$. Also note that since O is 0 to start with and m is a power of 2, the addition of O and m can be modelled as bitwise or of O and m for the purpose of efficiency. It can also be checked (using assertion-checking in `Moped`) that the $(31-i)^{th}$ bit is false before the i^{th} iteration, and thus the carry-bit is always 0, justifying our simplification. Note that intuitively, half the input bits (the first 16 bits) are leaked by the program. This intuition is confirmed by the results.

Comparison with prototype sqifc [23]. As **sqifc** provides an automated tool (rather than just a method), we ran the examples of Table 1 on **sqifc**. We consistently outperformed the tool (with **sqifc** timing out on several examples). However, we point that it is not exactly a fair comparison as we can guide the efficient computation of the summary by choosing the variable ordering, which has a considerable effect on our timings. The same optimization cannot be applied to **sqifc** because it is based on different concepts.

5 Leakage in probabilistic programs

We also generalized our algorithms for computing information leakage in programs that allow probabilistic choices. The *summary* of a probabilistic program is the *channel matrix*. The channel matrix on inputs S and outputs O is the $S \times O$ matrix such that its (s, o) entry is the conditional probability of observing o given s. More precisely, for a probabilistic program P with $\mathcal{G} = \{x_1, \ldots, x_n\}$ as the set of global variables, and $\mathcal{G}' = \{x'_1, \ldots, x'_n\}$ a set of distinct variables disjoint from $\mathcal{G}$, the *summary* of program P, denoted by T_P is the function $T_P : 2^{(\mathcal{G} \cup \mathcal{G}')} \to \mathbb{R}^+$ such that for every $z_1, \ldots, z_n, z'_1, \ldots, z'_n \in \{\mathtt{true}, \mathtt{false}\}$, $T_P(z_1, \ldots, z_n, z'_1, \ldots, z'_n)$ is the conditional probability that the programs outputs $(z'_1, \ldots, z'_n)$ given that the input to the program P is $(z_1, \ldots, z_n)$.

Just as the case for non-probabilistic programs, the summary relation for probabilistic programs can be computed using ADD-based fixed-point algorithms. Once again, we can give symbolic algorithms to compute the information leaked. We have implemented these symbolic algorithms in `Moped-QLeak` (currently we do not support restricted domains for probabilistic programs). Moped does not support probabilistic model-checking, so we also implemented the symbolic fixed-point algorithms for computing the summary also in Moped-QLeak. We used Moped-QLeak to compute information leakage in the dining cryptographer's problem. The symbolic algorithms and the results are discussed in detail in the longer version of the paper available at `http://people.cs.missouri.edu/~chadhar/mql/`.

6 Conclusions and future work

We gave symbolic algorithms for computing the information leaked by Boolean programs when information leakage is measured using min-entropy and Shannon entropy. The advantage of our approach is that these algorithms can be integrated with any BDD-based model checking tool that computes reachability in Boolean programs by computing program summaries. We made such an integration with Moped, with promising experimental results. The leakage calculations themselves add little overhead. The main limiting factor in these calculations seems to be the size of the OBDDs constructed in the computation. As is standard with symbolic approaches, the size of BDDs is sensitive to the variable ordering. Since Moped by itself does not compute the most efficient ordering (and puts the onus on the user), we sometimes had to rewrite our examples to achieve good performance. We also generalized our symbolic algorithms for computing information leakage in probabilistic programs. These algorithms have also been integrated in Moped.

In order to make symbolic model-checking more amenable to automation, many automated abstraction refinement techniques have been proposed in literature. We plan to investigate these techniques for our symbolic algorithms. In particular, we plan to integrate the counterexample guided abstraction-refinement framework in our symbolic algorithms. Currently, our implementation only supports non-recursive programs. However, the algorithms we presented for computing information leakage assume only that program summaries be computed. Thus, in principle, we can support programs that have both recursion and probabilistic choices, and we plan to extend support to such programs in future.

Acknowledgements. Rohit Chadha was supported by NSF grant CNS 1314338. Umang Mathur was supported in part by an INRIA student internship. Stefan Schwoon was supported by an INRIA *chaire d'excellence*.

References

1 S. B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, 27(6):509–516, 1978.

2 M. E. Andrés, C. Palamidessi, P. van Rossum, and G. Smith. Computing the leakage of information-hiding systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 373–389, 2010.

3 M. Backes, B. Köpf, and A. Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE Symposium on Security and Privacy*, pages 141–153, 2009.

4 F. Biondi, A. Legay, L. Traonouez, and A. Wasowski. Quail: A quantitative security analyzer for imperative code. In *Computer Aided Verification*, pages 702–707, 2013.

5 R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

6 R. Chadha, D. Kini, and M. Viswanathan. Quantitative information flow in boolean programs. In *Principles of Security and Trust*, pages 103–119, 2014.

7 K. Chatzikokolakis, T. Chothia, and A. Guha. Statistical measurement of information leakage. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 390–404, 2010.

8 K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Probability of error in information-hiding protocols. In *IEEE Computer Security Foundations Symp.*, pages 341–354, 2007.

9 K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2-4), 2008.

10 T. Chothia, Y. Kawamoto, C. Novakovic, and D. Parker. Probabilistic point-to-point information leakage. In *IEEE Computer Security Foundations Symp.*, pages 193–205, 2013.

11 D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

12 E. M. Clarke, K. L. Mcmillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. *Formal Methods in System Design*, 10(2-3):137–148, 1997.

13 J. Esparza, S. Kiefer, and S. Schwoon. Abstraction refinement with Craig interpolation and symbolic pushdown systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 3920, pages 489–503, 2006.

14 A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444, 2006.

15 V. Klebanov. Precise quantitative information flow analysis using symbolic model counting. 1st International Workshop on Quantitative Aspects of Security Assurance, 2012.

16 V. Klebanov, N. Manthey, and C. J. Muise. SAT-based analysis and quantification of information flow in programs. In *International Conference on Quantitative Evaluation of Systems*, pages 177–192, 2013.

17 B. Köpf and D. A. Basin. An information-theoretic model for adaptive side-channel attacks. In *ACM Conference on Computer and Communications Security*, pages 286–296, 2007.

18 B. Köpf and A. Rybalchenko. Approximation and randomization for quantitative information-flow analysis. In *IEEE Computer Security Foundations Symp.*, pages 3–14, 2010.

19 C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, 1959.

20 Z. Meng and G. Smith. Faster two-bit pattern analysis of leakage. 2nd International Workshop on Quantitative Aspects of Security Assurance, 2013.

21 Z. Meng and G. Smith. Calculating bounds on information leakage using two-bit patterns. In *Workshop on Programming Languages and Analysis for Security*, 2011.

22 J. K. Millen. Covert channel capacity. In *IEEE Symposium on Security and Privacy*, pages 60–66, 1987.

23 Q. Phan and P. Malacaria. Abstract model counting: a novel approach for quantification of information leaks. In *ACM Symposium on Information, Computer and Communications Security*, pages 283–292, 2014.

24 J. M. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems.* AMS, 2004.

25 G. Smith. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computation Structures*, pages 288–302, 2009.

26 P. Černý, K. Chatterjee, and T. A. Henzinger. The complexity of quantitative information flow problems. In *IEEE Computer Security Foundations Symposium*, pages 205–217, 2011.

27 H. Yasuoka and T. Terauchi. Quantitative Information Flow – Verification Hardness and Possibilities. In *IEEE Computer Security Foundations Symposium*, pages 15–27, 2010.

28 H. Yasuoka and T. Terauchi. Quantitative information flow as safety and liveness hyperproperties. In *Quantitative Aspects of Programming Languages and Systems*, pages 77–91, 2012.

Information Leakage of Non-Terminating Processes*

Fabrizio Biondi[1], **Axel Legay**[1], **Bo Friis Nielsen**[2], **Pasquale Malacaria**[3], and **Andrzej Wąsowski**[4]

1 **INRIA Rennes, France,** `{fabrizio.biondi,axel.legay}@inria.fr`
2 **Technical University of Denmark, Denmark,** `bfn@imm.dtu.dk`
3 **Queen Mary University of London,** `p.malacaria@qmul.ac.uk`
4 **IT University of Copenhagen, Denmark,** `wasowski@itu.dk`

Abstract

In recent years, quantitative security techniques have been providing effective measures of the security of a system against an attacker. Such techniques usually assume that the system produces a finite amount of observations based on a finite amount of secret bits and terminates, and the attack is based on these observations. By modeling systems with Markov chains, we are able to measure the effectiveness of attacks on non-terminating systems. Such systems do not necessarily produce a finite amount of output and are not necessarily based on a finite amount of secret bits. We provide characterizations and algorithms to define meaningful measures of security for non-terminating systems, and to compute them when possible. We also study the bounded versions of the problems, and show examples of non-terminating programs and how their effectiveness in protecting their secret can be measured.

1998 ACM Subject Classification D.4.6 Security and Protection, G.3 Stochastic Processes, H.1.1 Systems and Information Theory

Keywords and phrases Quantitative information flow, Markov chain, information leakage, infinite execution

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.517

1 Introduction

Information-theoretical quantitative security techniques evaluate the effectiveness of a system in protecting a secret it depends on. Given a known finite size of a secret in bits, they quantify how many bits of the secret can be inferred by an attacker able to observe the system's output. This value is referred to as *information leakage*, or just leakage. Leakage quantification techniques have been successfully applied to security problems, including proving the effectiveness of bug fixes to the Linux kernel [10], quantifying anonymity [6], and analyzing side channel attacks to the cache of a processor [11].

The theory behind these techniques commonly assumes that the program under analysis terminates at some point, and the computed leakage corresponds to the amount of information that the attacker gains at program termination time. When the secret does not change during computation, a program can be modeled as a *channel matrix*, assigning the conditional probability of each possible output for each possible input. The size of the channel matrix is

* The research presented in this paper has been partially supported by the MEALS project and by the MSR-INRIA project "Privacy-Friendly Services and Apps".

Secret	Observation	Leakage	Leakage rate	Reference
Finite	Finite	Finite	0	Paper [4]
Finite	Infinite	Finite	0	Section 3
Infinite	Finite	Finite	0	Section 3
Infinite	Infinite	Pot. infinite	Finite	Section 4

Figure 1 Leakage of various process topologies.

usually exponential in both secret and output size. We have previously proposed the use of Markovian models instead to overcome this problem [4].

Markovian models can also be conveniently used to model non-terminating processes, something finite-size channel matrices cannot do. This allows us to study leakage properties of systems like webservices, server modules and operating system daemons.

In this paper we provide techniques and algorithms to quantify the Shannon leakage and leakage rate of non-terminating processes. Shannon leakage has a clear operational significance related to the number of attempts that an attacker has to try to guess a secret [13]. Other measures exist, modeling different security properties (e. g. [16]). Our contributions are:

- We characterize program-attacker scenarios according to the finiteness of the system's secret and the finiteness of the attacker's observation. We show how this characterization influences the finiteness of the information leakage in the scenario.
- We provide a method to compute information leakage of a scenario with either an infinite secret or an infinite observation. Such scenarios cannot even be modeled with a finite size channel matrix. We demonstrate this method with an example.
- We provide a method to compute the rate of information leakage per time unit, when the leakage itself is infinite. This is the case when a scenario has an infinite secret and an infinite observation, as is common e. g. in webservices. We demonstrate this method using a mix node as an example.
- We provide an algorithm to compute how much information is leaked from a given time to another given time.
- We show that determining the exact time in which a given amount of information is leaked is hard, by reduction to the knowingly hard to decide Skolem's problem.

We distinguish four possible scenarios, according to whether the observation by the attacker is finite or infinite and whether the secret itself is finite or infinite. The cases are summarized in Fig. 1. The case with finite observation over a program depending on a finite secret is the terminating case we considered previously [4], while the others will be considered in this paper. When only one of observation or secret is finite the leakage is finite but cannot be computed using the method we introduced previously [4], thus we provide a new technique in Section 3. When both observation and secret are infinite, the leakage is potentially infinite. In this case we compute the rate of leakage, i. e. the amount of information leaked for each time unit. Intuitively, this quantifies the average amount of information the attacker infers for each time unit over an infinite time. This is presented in Section 4. In Section 5 we analyze how much information is leaked in a given time frame and how much time it takes to leak a given amount of information. Section 6 concludes the paper and discusses related work.

2 Background

We refer to literature [8] for the definitions of sample space S, probability of event $P(E)$ and so on. $\mathcal{X}$ is a *discrete stochastic process* if it is an indexed infinite sequence of discrete random variables $(X_0, X_1, X_2, \ldots)$. A discrete stochastic process is a *Markov chain* $\mathcal{C} = (C_0, C_1, C_2, \ldots)$ iff $\forall k \in \mathbb{N}.\ P(C_k|C_{k-1}, C_{k-2}, \ldots, C_1, C_0) = P(C_k|C_{k-1})$. A Markov chain on a sample space S can also be defined as follows:

▶ **Definition 1.** A tuple $\mathcal{C} = (S, s_0, P)$ is a Markov Chain (MC), if S is a finite set of states, $s_0 \in S$ is the initial state and P is a single $|S| \times |S|$ probability transition matrix, so $\forall s, t \in S.\ P_{s,t} \geq 0$ and $\forall s \in S.\ \sum_{t\in S} P_{s,t} = 1$.

The probability of transitioning from any state s to a state t in k steps can be found as the entry of index (s, t) in P^k [8]. We write $\pi^{(k)}$ for the probability distribution vector over S at time k and $\pi_s^{(k)}$ the probability of visiting the state s at time k; note that $\pi^{(k)} = \pi_0 P^k$, where $\pi_s^{(0)}$ is 1 if $s = s_0$ and 0 otherwise. A probability distribution $\bar{\pi}$ over the states of the chain is *stationary* if $\bar{\pi} = \bar{\pi}P$. Given an initial distribution $\pi^{(0)}$ we compute the unique stationary *limit distribution* μ as $\mu = \lim_{k\to\infty} \pi^{(0)} P^k$.

We write ξ_s for the *expected residence time* of state $s \in S$: $\xi_s = \sum_{k=0}^{\infty} P^k_{s_0,s}$. A state $s \in S$ is *absorbing* if $P_{s,s} = 1$. In the figures we do not draw the looping transition of the absorbing states, to reduce clutter.

We will enrich our Markovian models with a finite set $\mathtt{V}$ of natural-valued variables, and for simplicity we assume that there is a very large finite bit-size M such that a variable is at most M bit long. We define an assignment function $A : S \to [0, 2^M - 1]^{|\mathtt{V}|}$ assigning to each state the values of the variables in that state. We write $\mathtt{v}(s)$ to denote the value of the variable $\mathtt{v} \in \mathtt{V}$ in the state $s \in S$. Consider a stochastic process representing the value of a variable $\mathtt{v}$ over time, derived fro the behavior of a Markov chain labeled with valuations of this variable. We will call this process the *marginal process*, or just *marginal*, $\mathcal{C}_{|\mathtt{v}}$ on $\mathtt{v}$, formally:

▶ **Definition 2.** Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain and $\mathtt{v} \in \mathtt{V}$ a variable. Then we define the *marginal process* $\mathcal{C}_{|\mathtt{v}}$ *of* $\mathcal{C}$ *on* $\mathtt{v}$ as a stochastic process $(\mathtt{v}_1, \mathtt{v}_2, \ldots)$ where $\forall n.\ P(\mathtt{v}_k = n) = \sum_{\{s|\mathtt{v}(s)=n\}} \pi_s^{(k)}$

We will use $\mathtt{v}$ to denote the marginal process when it is clear from the context that we refer to it. Note that $\mathcal{C}_{|\mathtt{v}}$ is not necessarily a Markov chain. When it is, it can be drawn like in Fig. 3bcd. In the paper we will allow assignments of sets of values to variables and marginals on sets of variables; such extensions are straightforward, since multiple variables can be seen as a single variable on their product space. Assume that the system modeled by $\mathcal{C}$ has a single secret variable $\mathtt{h}$ and a single observable variable $\mathtt{o}$. Then the distributions over the marginal processes $\mathcal{C}_{|\mathtt{h}}$ and $\mathcal{C}_{|\mathtt{o}}$ model the behavior of the secret and observable variable respectively at each time step, and their correlation quantifies the amount of information about the secret that can be inferred by observing the observable variable.

Entropy is a measure of the uncertainty of a probability distribution. The following definitions are standard:

▶ **Definition 3** ([8]). Let X and Y be two random variables with probability mass functions $p(x)$ and $p(y)$ respectively and joint probability mass function $p(x, y)$. Then we define the following non-negative real-valued functions:

- Entropy $H(X) = -\sum_{x\in X} p(x) \log_2 p(x)$
- Joint entropy $H(X, Y) = -\sum_{x\in X} \sum_{y\in Y} p(x, y) \log_2 p(x, y)$

- Conditional entropy $H(X|Y) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(x|y) =$
  $= \sum_{y\in Y} p(y)H(X|Y=y) = -\sum_{y\in Y} p(y)\sum_{x\in X} p(x|y)\log_2 p(x|y) =$
  $= H(X,Y) - H(Y)$ (chain rule)
- Mutual information $I(X;Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2\left(\frac{p(x,y)}{p(x)p(y)}\right) =$
  $= H(X) + H(Y) - H(X,Y) \leq \min(H(X), H(Y))$

Mutual information can be generalized to two vectors of random variables $\bar{X}, \bar{Y}$ as $I(\bar{X};\bar{Y}) = \sum_{\bar{x}\in\bar{X}}\sum_{\bar{y}\in\bar{Y}} p(\bar{x},\bar{y})\log_2\left(\frac{p(\bar{x},\bar{y})}{p(\bar{x})p(\bar{y})}\right)$.

▶ **Definition 4.** [8] Let $\mathcal{X} = (X_1, X_2, \ldots)$ and $\mathcal{Y} = (Y_1, Y_2, \ldots)$ be two stochastic processes. Then we define the following non-negative real-valued functions:

- Entropy $H(\mathcal{X}) = \lim_{k\to\infty} H(X_1, X_2, \ldots, X_k)$
- Entropy rate $\bar{H}(\mathcal{X}) = \lim_{k\to\infty}\frac{1}{k}H(X_1, X_2, \ldots, X_k)$ when the limit exists
- Mutual information $I(\mathcal{X};\mathcal{Y}) = \lim_{k\to\infty} I(X_1, X_2, \ldots, X_k; Y_1, Y_2, \ldots, Y_k)$
- Mutual information rate $\bar{I}(\mathcal{X};\mathcal{Y}) = \lim_{k\to\infty}\frac{1}{k}I(X_1, X_2, \ldots, X_k; Y_1, Y_2, \ldots, Y_k)$ when the limit exists

Entropy and mutual information of stochastic processes always exist, as shown in Section 3. Entropy rate and mutual information rate may not exist in general, but exist when the stochastic processes are Markov chains [8]; we discuss them in Section 4.

Since every state s in a MC $\mathcal{C} = (S, s_0, P)$ has a discrete probability distribution over the successor states we can calculate the entropy of this distribution, the *local entropy*:

▶ **Definition 5.** Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain. Then for each state $s \in S$ we define the *local entropy* of s as $L(s) = -\sum_{t\in S} P_{s,t}\log_2 P_{s,t}$

Note that $L(s) \leq \log_2(|S|)$ [5]. If a stochastic process is a Markov chain $\mathcal{C}$, its entropy $H(\mathcal{C})$ can be computed by considering the local entropy $L(s)$ as the expected reward of a state s and then computing the expected total reward of the chain [5]: $H(\mathcal{C}) = \sum_{s\in S} L(s)\xi_s$. It is also known that the entropy rate can be computed similarly by summing the local entropies of each state weighted by the state's probability in the limiting distribution [8]: $\bar{H}(\mathcal{C}) = \sum_{s\in S} L(s)\mu_s$.

In this paper we use information theory to compute the amount of bits of a secret variable `h` that can be inferred by an attacker able to observe the value of an observable variable `o` at any moment in time. We call this amount *Shannon leakage* or just leakage, and it corresponds to the mutual information $I(\mathcal{C}_{|\texttt{o}}, \mathcal{C}_{|\texttt{h}})$ between the marginal on the secret and the marginal on the observable variable.

Operationally, Shannon leakage is related to the number of attempts that an attacker has to do to guess the value of the secret. Other leakage measures exist, but Shannon leakage is the only one for which the chain rule of Definition 3 holds; since the chain rule is used in many results in this work, we do not expect such results to extend to other leakage measures.

The modeling of a process as a Markov chain in our context starts by dividing the variables in private and public variables. Private variables, including the secret variable `h`, are the ones whose value is not defined at compilation time. In each state of the Markov chain a set of allowed values is assigned to each private variable. Public variables, including the observable variable `o` and the program counter `pc`, are variables whose value is known to the analyst. On each state a given value is assigned to each public variable.

Given the source code and a prior distribution over the private variables, we have enough information to build a Markov chain representing the semantics, since for each state we can determine its successor states and the corresponding transition probabilities. We show a

```
secret int1 h;
observable int1 o;
public int1 r;
random r := randombit(0.75);
assign o := h ^ r;
return;
```

Figure 2 Bit XOR example: source code.

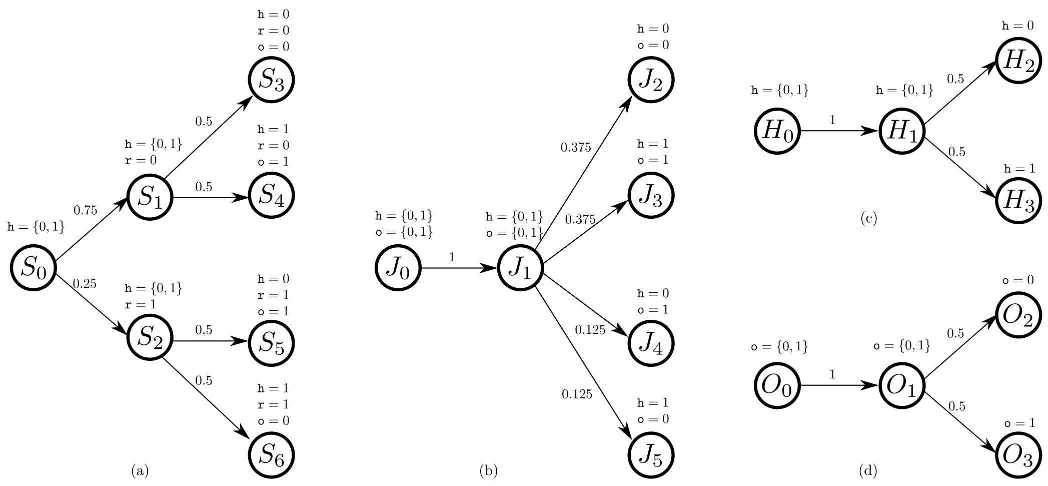


Figure 3 Bit XOR example: a) Markov chain semantics $\mathcal{C}$. b) Joint marginal $\mathcal{C}_{|(o,h)}$. c) Secret's marginal $\mathcal{C}_{|h}$. d) Observer's marginal $\mathcal{C}_{|o}$.

simple example, and refer to [4] for the complete semantics. The source code for the example is shown in Fig. 2 and the corresponding Markov chain semantics in Fig. 3a.

Let `h` be a secret bit, `o` an observable bit and `r` a random bit being assigned the value 0 with probability 0.75 and 1 otherwise. We assign to `o` the result of the exclusive OR between `h` and `r` and terminate. We want to quantify the amount of information about `h` that can be inferred by knowing the value of `o`.

To compute the leakage we need to compute three marginals from the Markov chain semantics:

Joint marginal The joint marginal process $\mathcal{C}_{|(o,h)}$ models the joint behavior of the secret and observable variables. It is shown in Fig. 3b.

Secret's marginal The secret's marginal process $\mathcal{C}_{|h}$ models the behavior of the secret variable. It is shown in Fig. 3c.

Observer's marginal The observer's marginal process $\mathcal{C}_{|o}$ models the behavior of the observable variable. It is shown in Fig. 3d.

Finally we compute the mutual information between the secret and observable variable using the formula $I(X;Y) = H(X) + H(Y) - H(X,Y)$, obtaining $I(\mathtt{o};\mathtt{h}) = I(\mathcal{C}_{|o};\mathcal{C}_{|h}) = H(\mathcal{C}_{|o}) + H(\mathcal{C}_{|h})) - H(\mathcal{C}_{|(o,h)})) = 1 + 1 - 1.8112\ldots \approx 0.1887$ bits, proving that the program leaks ≈ 0.1887 bits, or 18.87% of the secret.

3 Non-terminating Processes with Finite Leakage

The Markov chain semantics of the system describes the joint behavior of all variables. To compute information leakage we are only interested in the secret and the observable variables, so we can restrict to them only for simplicity. We assume that the system has a single secret variable $\mathtt{h}$ with uniform prior distribution and a single observable variable $\mathtt{o}$, but the procedure does not change for multiple secret or observable variables. We remark that, even though the attacker can perform multiple observations, we do not model the case in which the attacker actually interacts with the system. In such case directed information would have to be used as the leakage metric instead of mutual information. We refer to Alvim et al. [2] for the details.

The behavior of the secret variable $\mathtt{h}$ is modeled by the marginal $\mathcal{C}_{|\mathtt{h}}$, and similarly the behavior of $\mathtt{o}$ is modeled by $\mathcal{C}_{|\mathtt{o}}$ and the joint behavior of the two variables by $\mathcal{C}_{|\mathtt{o},\mathtt{h}}$. The following lemma shows the existence of the entropy values of such marginals and a sufficient condition for the finiteness of their mutual information:

▶ **Theorem 6.** *Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain with secrets and observations and $\mathcal{C}_{|\mathtt{o}}$ and $\mathcal{C}_{|\mathtt{h}}$ its marginals on the observable and secret variables, respectively. Then:*
- *$H(\mathcal{C}_{|\mathtt{o}})$, $H(\mathcal{C}_{|\mathtt{h}})$ and $I(\mathcal{C}_{|\mathtt{o}};\mathcal{C}_{|\mathtt{h}})$ exist;*
- *$H(\mathcal{C}_{|\mathtt{o}}) < \infty \vee H(\mathcal{C}_{|\mathtt{h}}) < \infty \Rightarrow I(\mathcal{C}_{|\mathtt{o}}, \mathcal{C}_{|\mathtt{h}}) < \infty$*

Intuitively, if $H(\mathcal{C}_{|\mathtt{o}}) < \infty$ and $H(\mathcal{C}_{|\mathtt{h}}) = \infty$ then there is an infinite number of secret bits but only a finite amount of observations we can analyze. In the opposite case where $H(\mathcal{C}_{|\mathtt{o}}) = \infty$ and $H(\mathcal{C}_{|\mathtt{h}}) < \infty$ we can analyze an infinite number of observations, but there is only a finite amount of secret bits to be discovered.

If either $H(\mathcal{C}_{|\mathtt{o}}) = \infty$ or $H(\mathcal{C}_{|\mathtt{h}}) = \infty$ then $H(\mathcal{C}_{|\mathtt{o},\mathtt{h}}) = \infty$, since $H(X,Y) \geq \max(H(X), H(Y))$. It follows that if either the observation or the secret are infinite but not both, the formula $I(\mathcal{C}_{|\mathtt{o}}, \mathcal{C}_{|\mathtt{h}}) = H(\mathcal{C}_{|\mathtt{o}}) + H(\mathcal{C}_{|\mathtt{h}}) - H(\mathcal{C}_{|\mathtt{o},\mathtt{h}})$ will produce an indeterminate form $\infty - \infty$ and thus cannot be directly used to compute the leakage. Nonetheless, in both cases leakage has a finite value by Theorem 6.

If any marginal is a Markov chain, it is possible to compute its entropy in polynomial time in the size of the chain [5]. Otherwise, consider that the entropies of the marginals are limit computations, since $H(\mathcal{C}_{|\mathtt{v}}) = \lim_{k\to\infty} H(\mathtt{v}_1, \ldots, \mathtt{v}_k)$. This allows us to compute mutual information using the limit of the entropies of the marginal processes:

$$\begin{aligned} I(\mathcal{C}_{|\mathtt{o}};\mathcal{C}_{|\mathtt{h}}) &= \lim_{k\to\infty} I(\mathtt{o}_1, \ldots, \mathtt{o}_k; \mathtt{h}_1, \ldots, \mathtt{h}_k) \\ &= \lim_{k\to\infty} (H(\mathtt{o}_1, \ldots, \mathtt{o}_k) + H(\mathtt{h}_1, \ldots, \mathtt{h}_k) - H((\mathtt{o},\mathtt{h})_1, \ldots, (\mathtt{o},\mathtt{h})_k)) \end{aligned}$$

The limit above computes information leakage in any case, but is not always the most efficient option available. When it is known that the secret (resp. observation) is finite, it is more efficient to use the formula $I(\mathcal{C}_{|\mathtt{o}};\mathcal{C}_{|\mathtt{h}}) = H(\mathcal{C}_{|\mathtt{h}}) - H(\mathcal{C}_{|\mathtt{h}}|\mathcal{C}_{|\mathtt{o}})$ (resp. $I(\mathcal{C}_{|\mathtt{o}};\mathcal{C}_{|\mathtt{h}}) = H(\mathcal{C}_{|\mathtt{o}}) - H(\mathcal{C}_{|\mathtt{o}}|\mathcal{C}_{|\mathtt{h}})$). Remember that when both secret and observation are finite the process terminates, so the procedure we proposed in [4] can be used with some additional assumptions.

3.1 Example: A Non-terminating Program on a Finite Secret

We now solve a case in which the secret is finite and Markovian and the observation infinite. Consider a program (Figure 4) with a secret bit $\mathtt{h}$. If $\mathtt{h}$ is 0 the program produces an infinite string of zeroes and ones with the same probability 0.5, starting with a zero. If $\mathtt{h}$ is 1 the program also produces a string of zeroes and ones starting with a zero, but the probability that it will produce a zero is 0.75. Note that this program cannot be encoded as a finite channel matrix, as it has an infinite amount of possible outputs.

An attacker may be able to observe this infinite string and infer information about the secret by studying the frequencies of zeroes and ones. The attacker starts with no

```
secret int1 h;
if (h==0) then
  observable int1 o:=0;
  while (0==0) do
    random o:=randombit
        (0.5);
  od
else
  observable int1 o:=0;
  while (0==0) do
    random o:=randombit
        (0.75);
  od
fi
return;
```

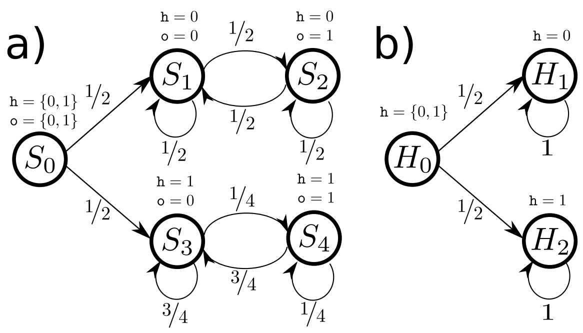


Figure 4 Non-terminating leaking program example. On the left: program code. On the right: Markov chain semantics a) joint marginal $\mathcal{C}_{|o,h}$ b) secret's marginal $\mathcal{C}_{|h}$.

knowledge of the secret, which is encoded as an initial uniform distribution over the secret bit h. Reasonably, an attacker observing the output for an infinite time would be able to decide whether the frequency of zeroes is 0.5 or 0.75 and infer the value of h consequently. The Markov chain semantics for it is shown in Fig. 4a on the right.

Since $\mathtt{h}_1 = \mathtt{h}_2 = \mathtt{h}_3 = \ldots$ we will just call it h. The behavior of h is modeled by the Markov chain in Fig. 4b on the right, and its entropy is $H(\mathtt{h}) = 1$ bit. $H(\mathcal{C}_{|\mathtt{h}}|\mathcal{C}_{|\mathtt{o}})$ corresponds to $\lim_{k\to\infty} H(\mathtt{h}|\mathtt{o}_1, \ldots, \mathtt{o}_k)$.

We compute $H(\mathtt{h}|\mathtt{o}_1, \ldots, \mathtt{o}_k)$ for $k \to \infty$. Note that at time 1 o is always 0, then it changes randomly depending on the value of h. We will write down the joint distribution of h and o as a function of k and use it to compute the marginal over o and finally the conditional entropy.

The joint distribution of h and o is shown in the Appendix due to space constraints. Now let $w^k \in \{0,1\}^k$ be a sequence of k bits. Consider the formula for conditional entropy:

$$H(\mathtt{h}|\mathtt{o}_1, \ldots, \mathtt{o}_k) = \sum_{w^k \in \{0,1\}^k} P(\mathtt{o}_1, \ldots, \mathtt{o}_k = w^k) H(\mathtt{h}|\mathtt{o}_1, \ldots, \mathtt{o}_k = w^k) \tag{1}$$

In our case it holds that $H(\mathtt{h}|\mathtt{o}) = \lim_{k\to\infty} H(\mathtt{h}|\mathtt{o}_1, \ldots, \mathtt{o}_k) = 0$ thus $I(\mathtt{o};\mathtt{h}) = H(\mathtt{h}) - H(\mathtt{h}|\mathtt{o}) = 1 - 0 = 1$ bit. The leakage of the program in Fig. 4 is 1 bit, proving that an attacker able to analyze the bit streams produced by the system will eventually learn the value of the secret h with an arbitrary confidence. Note that this considers an attacker able to observe the system for an infinite time.

More importantly, note that the marginal process of o is *not* a Markov chain. This depends on the fact that the joint distribution depends also on the information that the attacker has about h, so while the attacker gathers information about o and h the joint distribution changes and thus the marginal distribution of o changes also. Nonetheless, the marginal process can be represented in a closed form like the one in Fig. 4.

4 Leakage Rate of a Markov Chain

In the case in which $H(\mathcal{C}_{|\mathtt{o}}) = \infty$ and $H(\mathcal{C}_{|h}) = \infty$, i. e. when the secret is an infinite number of bits and the observer can observe it for an infinite time, then the leakage $I(\mathtt{o},\mathtt{h})$ can be infinite. In this case it is more interesting to compute how much information the process

```
   Data: A Markov Chain $\mathcal{C} = (S, s_0, P)$ and its initial probability distribution $\pi^{(0)}$.
   Result: The limit probability distribution $\mu$ of the chain.
1  foreach transient state $t$ do
2      $\mu_t \leftarrow 0$;
3      foreach $u \in S \setminus \{t\}$ do
4          $\pi^{(0)}_u \leftarrow \pi^{(0)}_u + \pi^{(0)}_t \frac{P_{t,u}}{1-P_{t,t}}$
5          foreach $s \in S \setminus \{t\}$ do
6              $P_{s,u} \leftarrow P_{s,u} + P_{s,t}\frac{P_{t,u}}{1-P_{t,t}}$
7              $P_{s,t} \leftarrow 0$
8          end
9      end
10 end
11 foreach end component $R_i$ do
12     Let $\pi^{(\infty)}_{R_i} = \sum_{r \in R_i} \pi^{(0)}_s$ and $E_i$ be a system of linear equations;
13     foreach $r \in R_i$ do  Add to $E_i$ the equation $\mu_r = \sum_{r' \in R_i} \mu_{r'} P_{r',r}$ ;
14     Solve the system E' under the condition $\sum_{r \in R_i} \mu_r = \pi^{(\infty)}_{R_i}$ to obtain $\mu_r$ for each
       $r \in R_i$;
15 end
```

Algorithm 1. Compute the limit distribution of a Markov chain.

leaks for each time step. This quantity is known as *leakage rate*, and corresponds to the *mutual information rate* of the secret and observable.

Note that the computation of leakage as a rate over time assumes that the attacker is able to keep track of the discrete time, so in this section we will assume that every constant-time operation takes 1 time step. This can be equivalently stated as saying that all transitions between states of the Markov chain semantics represent observable steps.

To compute leakage rate, we encode the process-attacker scenario with a Markov chain as shown in Section 2 and compute the joint, secret and attacker's marginal, which may not be Markovian. Then we can use the marginals to compute leakage rate by applying the following definition:

▶ **Definition 7.** Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain and $\mathcal{C}_{|\texttt{o},\texttt{h}}$, $\mathcal{C}_{|\texttt{o}}$ and $\mathcal{C}_h$ its marginals on `(o,h)`, `o` and `h` respectively. Then the *leakage rate* $\bar{I}$ is defined as $\bar{I}(\mathcal{C}_{|\texttt{o}};\mathcal{C}_h) = \bar{H}(\mathcal{C}_{|\texttt{o}}) + \bar{H}(\mathcal{C}_{|\texttt{h}}) - \bar{H}(\mathcal{C}_{|\texttt{o},\texttt{h}})$.

Leakage rate can also be computed as a limit, since

$$\begin{aligned}\bar{I}(\mathcal{C}_{|\texttt{o}};\mathcal{C}_{|\texttt{h}}) &= \lim_{k\to\infty} \frac{I(\texttt{o}_1,\ldots,\texttt{o}_k;\texttt{h}_1,\ldots,\texttt{h}_k)}{k} \\ &= \lim_{k\to\infty} \frac{(H(\texttt{o}_1,\ldots,\texttt{o}_k) + H(\texttt{h}_1,\ldots,\texttt{h}_k) - H((\texttt{o},\texttt{h})_1,\ldots,(\texttt{o},\texttt{h})_k))}{k}\end{aligned}$$

when the limit exists.

Generally both entropy and leakage rate could be infinite, for instance for a program that leaks 1 bit at time 1, 2 bits at time 2, and so on, the leakage rate would be infinite. Since we postulated that there exists a very large but finite maximum size M for the variables declared in the system, it is impossible to declare an unbounded amount of secret or observable bits on each step of the program execution. We do not think that this restriction limits significantly the programs that can be analyzed, while guaranteeing that the entropy and leakage rate do not diverge to positive infinity is a significantly useful result.

Both entropy rate and leakage rate may still oscillate, even though since they are defined in terms of Cesàro limits this happens only in pathological cases. We do not expect these cases to be common in normal secret-dependent systems, and leave finding a meaningful measure of leakage for these cases an open problem. This reflects similar issues in related definitions of leakage rate [7, 12].

A case in which both entropy and leakage rate exist is when the marginal processes modeling the behavior of the observable and secret variables are both Markovian. Intuitively, this happens when the secret gets periodically replaced with a new one, and thus the information the attacker has on it is reset to the prior information. We will show this with an example in Section 4.1.

When any of the marginals is a Markov chain it is possible to compute its entropy rate efficiently as $\bar{H}(\mathcal{C}) = \sum_{s\in S} L(s)\mu_s$, where μ is the limit distribution of the chain. The entropy rate of a Markov chain with a given initial probability distribution $\pi^{(0)}$ exists and is unique [8].

Computing the limit distribution can be accomplished on irreducible Markov chains by solving a system of linear equations, but the Markov chains we consider are usually reducible, so a new algorithm is required. Algorithm 1 computes the limit distribution of any Markov Chain $\mathcal{C} = (S, s_0, P)$. The algorithm uses the well-established concept of end components of a MC; each end component R_i behaves as an irreducible MC, so it is sufficient to compute the probability $\pi^{(\infty)}_{R_i}$ of eventually visiting R_i and redistribute $\pi^{(\infty)}_{R_i}$ among the states of R_i by solving a system of linear equations.

▶ **Theorem 8.** *Algorithm 1 terminates in polynomial time in $|S|$ and when it does it returns the limit distribution μ.*

Due to space constraints we refer to the Appendix for the proof of this theorem and a full explanation of the steps of Algorithm 1.

Since computing the local entropy of each state in time $O(|S|^2)$ is trivial, the formula $\bar{H}(\mathcal{C}) = \sum_{s\in S} L(s)\mu_s$ can be used to compute entropy rate of a Markov chain in polynomial time. Note that this is a particular case of the computation of an expected infinite-horizon reward rate of a reward function defined on the transitions of a Markov chain.

Having computed the entropy rates of the joint, secret and attacker's marginals we can apply Definition 7 to obtain the leakage rate of the system.

4.1 Example: Leaking Mix Node Implementation

We show an example of a program leaking an infinite amount of information and we compute its leakage rate using the method described above.

A mix node [9] is a program meant to scramble the order in which packages are routed through a network, to increase the anonymity of the sender. Even if the packages are encrypted, some information about the sender could be inferred by observing the order in which they are forwarded. A mix node changes this order to a random one, thus making it harder for an attacker to connect each package to its sender.

A mix node waits until it has accumulated a fixed amount of packages, and then forwards them in a random order. If the exit order of the packages is independent from the entrance order, then no information about the latter can be inferred by observing the former. We will present an implementation of a mix node where the entrance and exit order are not independent and compute the rate of the information leakage.

```
 1 secret int3 inorder;
 2 public int3 rand;
 3 observable int3 outorder;
 4 while (0==0) do
 5   assign inorder := [0,5];
 6   random rand := random(0,5);
 7   assign outorder :=
 8       (inorder ^ rand)%6;
 9 od
10 return;
```

Figure 5 A leaking implementation of a mix node.

The implementation of the mix node is shown in Fig. 5. This particular node waits until it has accumulated 3 packages and then sends them in a random order. Naming the packages A, B and C there are 6 possible entrance orders: ABC, ACB, BAC, BCA, CAB, and CBA. We will number them from 0 to 5.

In line 5 of the code a random number from 0 to 5 is assigned to the secret variable `inorder`, modeling the secret entrance order. Then in line 6 a random value uniformly distributed from 0 to 5 is assigned to the variable `rand`. Finally, in line 7 the bitwise exclusive OR modulo 6 of the variables `inorder` and `rand` is assigned to the observable variable `outorder`, which represents the order in which the packages exit from the mix node and is observable to the attacker. After producing an exit order the mix node receives three more packages in a new entrance order, scrambles them the same way and forwards them, and so on forever.

Assume that the prior distribution over the input order is uniform. The resulting probability distribution on the exit order is $P(\texttt{outorder}) = \{0 \mapsto {}^{5}\!/_{18}, 1 \mapsto {}^{5}\!/_{18}, 2 \mapsto {}^{2}\!/_{18}, 3 \mapsto {}^{2}\!/_{18}, 4 \mapsto {}^{2}\!/_{18}, 5 \mapsto {}^{2}\!/_{18}\}$. This depends on the fact that bitwise OR and modulo operations do not preserve distribution uniformity.

The Markov chain semantics of the system has more than 300 states. It can be computed and analyzed automatically in less than a second. Assuming that each line of code is executed in one time step, the entropy rates of the marginals are $\bar{H}(\texttt{inorder}) = 0.86165\ldots$ bits, $\bar{H}(\texttt{outorder}) = 0.82766\ldots$ and $\bar{H}(\texttt{inorder}, \texttt{outorder}) = 1.59367\ldots$, giving a leakage rate of $\bar{I}(\texttt{inorder}; \texttt{outorder}) = \bar{H}(\texttt{inorder}) + \bar{H}(\texttt{outorder}) - \bar{H}(\texttt{inorder}, \texttt{outorder}) = 0.86165\ldots + 0.82766\ldots - 1.59367\ldots \approx 0.09564$ bits.

The leakage rate for each time unit is ≈ 0.09564 bits. Since the entropy rate of the secret is ≈ 0.86165 bits, we can conclude that this implementation of a mix node has a rate of leakage $0.09564/0.86165 \approx 11.1\%$ of each of its infinite secrets.

Note that in this simple case the loop always takes 3 time units to complete, so it would have been possible to just compute the leakage of one loop and divide it by 3, but in general loops do not compute for a fixed number of time units, e. g. if they contain multiple `return` statements.

5 Bounded Time/Leakage Analysis

We consider two similar bounded approaches to the leakage problem: computing the leakage of a Markov chain within a given time frame, or computing how long it takes for the Markov chain to leak a given amount of information.

```
Data: A Markov Chain $\mathcal{C} = (S, s_0, P)$ with the variables o and h, two integers $t_1$ and
      $t_2$ satisfying $t_1 \leq t_2$.
Result: The leakage from time $t_1$ to time $t_2$ $I^{(t_1,t_2)}(\mathtt{o},\mathtt{h})$
1  for $x \in \{\mathtt{o},\mathtt{h},(\mathtt{o},\mathtt{h})\}$ do
2      Compute the marginal $\mathcal{C}_{|x}$, and let $\pi_{|x}^{(t_1)}$ be the probability distribution over its
       states at time $t_1$ and $H^{(t_1)}(\mathcal{C}_{|x}) = H(\pi_{|x}^{(t_1)})$;
3  end
4  Compute $I^{(t_1,t_1)} = H^{(t_1)}(\mathcal{C}_{|\mathtt{o}}) + H^{(t_1)}(\mathcal{C}_{|\mathtt{h}}) - H^{(t_1)}(\mathcal{C}_{|\mathtt{o},\mathtt{h}})$;
5  for $i = t_1 + 1$ to $t_2$ do
6      for $x \in \{\mathtt{o},\mathtt{h},(\mathtt{o},\mathtt{h})\}$ do
7          $H^{(i)}(\mathcal{C}_{|x}) \leftarrow H^{(i-1)}(\mathcal{C}_{|x}) + \sum_{s \in S_{|x}} \pi_{|x}^{(i-1)}(s) L_{|x}(s)$;
8          $\pi_{|x}^{(i)} \leftarrow \pi_{|x}^{(i-1)} P_{|x}^{(i-1,i)}$;
9      end
10     $I^{(t_1,i)} \leftarrow H^{(i)}(\mathcal{C}_{|\mathtt{o}}) + H^{(i)}(\mathcal{C}_{|\mathtt{h}}) - H^{(i)}(\mathcal{C}_{|\mathtt{o},\mathtt{h}})$;
11 end
12 return $I^{(t_1,t_2)}$;
```

Algorithm 2. Compute the leakage of a MC from a time t_1 to a time t_2.

5.1 Bounded Time

We want to compute the leakage for an attacker that is able to observe the behavior of the program for $t < \infty$ time units. We abstract time by considering each time unit as a step in the evolution of the Markov chain modeling the system. The definition of mutual information from a time t_1 to a time $t_2 > t_1$ is as follows:

▶ **Definition 9.** Let $\mathcal{X}$ and $\mathcal{Y}$ be two stochastic processes. Then the mutual information between X_i and Y_i from time t_1 to time t_2 is $I(X_{t_1},\ldots,X_{t_2};Y_{t_1},\ldots,Y_{t_2}) = H(X_{t_1},\ldots,X_{t_2}) + H(Y_{t_1},\ldots,Y_{t_2}) - H(X_{t_1},\ldots,X_{t_2},Y_{t_1},\ldots,Y_{t_2})$

We will refer to it as $I^{(t_1,t_2)}(\mathcal{X};\mathcal{Y})$ for simplicity.

Consider as usual the Markov chain semantics $\mathcal{C} = (S, s_o, P)$ modeling the behavior of the system. We present an iterative algorithm to compute $I^{(t_1,t_2)}(\mathtt{o},\mathtt{h})$ in time $O(t_2|S|^2)$: the algorithm first computes the distribution at time t_1 and then computes the behavior of the chain until time t_2 while keeping track of the amount of leakage accumulated. In the algorithm let $S_{|x}$ be the state space of the marginal, $\pi_{|x}^{(i)}$ the distribution on the marginal at time i, and $P_{|x}^{(i,j)}$ the probability of transitioning from i to j in the marginal.

▶ **Theorem 10.** *Algorithm 2 terminates in time $O(t_2|S|^2)$ and when it does it outputs $I^{(t_1,t_2)}(o,h)$.*

Note that Algorithm 2 is pseudopolynomial, as it depends not only on the size of the chain but also on the parameter t_2. Also note that due to the Markov property it holds that $I^{(t_1,t_2)} = I^{(t_1',t_2')}$ whenever $\pi_{|\mathtt{o},\mathtt{h}}^{(t_1)} = \pi_{|\mathtt{o},\mathtt{h}}^{(t_1')}$ and $t_2 - t_1 = t_2' - t_1'$.

5.2 Bounded Leakage

We want to determine how many time units it takes for the system to leak a given amount c of bits of information. The problem is more complex than the one analyzed in the previous section, since leakage is a complex function of the behavior of the system in time and finding a way to bound or reverse it is not obvious.

We start by considering the qualitative version of the problem: does there exist a time t such that $I_t(\mathcal{C}_O, \mathcal{C}_h) \geq c$? To answer we note that the the sequence of leakages is monotonic non-decreasing over time, so if the answer is yes then the leakage will remain greater than c for each time $t' \geq t$. This allows us to answer the qualitative question by computing the leakage on the infinite time horizon as shown in Section 3; let it be l^∞. If $l^\infty < c$ then there is no time t such that the leakage is c, while if $l^\infty > c$ then such time exists. If $l^\infty = c$ then the system leaks c bits on the infinite time horizon but we have no guarantee that this amount will be reached in finite time.

In the case in which $l^\infty \geq c$ we can ask the quantitative question, i. e. at what time t the system will have leaked at least c bits. If $l^\infty > c$ we know that such time t exists, while if $l^\infty = c$ it may not. We will define the *bounded leakage problem* as follows: given a Markov chain $\mathcal{C} = (S, s_0, P)$ labeled with secrets and observations and a positive real number c, determine if there exists a finite time t such that the information leakage of the chain at time t is exactly c.

The problem is harder than it seems. For deterministic programs, it has been shown by Terauchi that it is not a k-safety property for any k [17]. The problem has also been addressed computationally by Heusser and Malacaria [10]. For randomized programs, we will show that the problem can be reduced from Skolem's problem [14]. While smaller instances have been shown to be decidable, the full decidability of Skolem's problem is still an open question [15]. Akshay et al. [1] show that Skolem's problem is equivalent to the following: given a Markov chain $\mathcal{C} = (S, s_0, P)$, a state s and a probability r determine whether there is a time t such that $\pi_s^{(t)} = r$. We will call this *Skolem's Markov chain reachability problem.* Intuitively, information leakage is a harder problem than reachability, as formally stated by the following theorem:

▶ **Theorem 11.** *Let $\mathbb{A}$ be an algorithm deciding the bounded leakage problem. Then $\mathbb{A}$ decides Skolem's Markov chain reachability problem.*

6 Conclusions and Related Work

We have shown how to provide meaningful measures of the effectiveness of a secret-dependent non-terminating program in protecting its secret, by computing Shannon leakage when its value is finite and Shannon leakage rate otherwise. Operationally, Shannon leakage is related to the expected number of guesses it will take for the attacker to find out the secret's value, so leakage and leakage rate can be used to understand the amount of time that the attacker will require to infer the system's secret [13]. To the same aim, we provided an algorithm that computes the amount of leakage from a program in a given time frame. Finally, we have shown that a precise quantification of the time required to leak a given amount is a hard problem, proving the complexity of the problem.

The quantification of leakage for an infinite observation and a finite or infinite secret has been recently considered by Chothia et al. [7]. Their framework is different from ours as they study probabilistic "point to point" leakage; also they provide no algorithms to compute leakage. The authors do not consider the case in which the observation is finite and the secret infinite. Also, in the infinite secret and observation case they explicitly do not consider the leakage rate per time unit, preferring to compute the leakage for each occurrence of the `secret` command in their framework.

Alvim et al. [2] study the setting of interactive systems, where secrets and observables can alternate during the computation and influence each other. They show that in this case mutual information is only an upper bound on leakage and "direct information" is a more precise leakage measure. This work is related to ours in that it investigates multi-stage processes but it presents significant differences as it doesn't investigate infinite leakage nor Markovian processes.

Recently Backes et al. also present a method for leakage rate computation based on stationary distribution of Markov chains, which they compute using PageRank [3]. We expect that Algorithm 1 would be a useful addition to their approach.

References

1 S. Akshay, Joël Ouaknine, Timos Antonopoulos, and James Worrell. Reachability problems for Markov chains. Personal communication, November 2013.

2 Mário S. Alvim, Miguel E. Andrés, and Catuscia Palamidessi. Quantitative information flow in interactive systems. *Journal of Computer Security*, 20(1):3–50, 2012.

3 Michael Backes, Goran Doychev, and Boris Köpf. Preventing side-channel leaks in web traffic: A formal approach. In *NDSS*. The Internet Society, 2013.

4 Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wasowski. Quantifying information leakage of randomized protocols. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *VMCAI*, 2013.

5 Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. In Adrian Horia Dediu and Carlos Martín-Vide, editors, *LATA*, 2013.

6 Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Inf. Comput.*, 206(2-4):378–401, 2008.

7 Tom Chothia, Yusuke Kawamoto, Chris Novakovic, and David Parker. Probabilistic point-to-point information leakage. In *CSF*, pages 193–205. IEEE, 2013.

8 T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2012.

9 George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *IEEE Symposium on Security and Privacy*, pages 2–15. IEEE, 2003.

10 Jonathan Heusser and Pasquale Malacaria. Quantifying information leaks in software. In C. Gates, M. Franz, and J. P. McDermott, editors, *ACSAC*, pages 261–269. ACM, 2010.

11 Boris Köpf, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. In P. Madhusudan and S. A. Seshia, editors, *CAV*, pages 564–580. Springer, 2012.

12 Pasquale Malacaria. Assessing security threats of looping constructs. In Martin Hofmann and Matthias Felleisen, editors, *POPL*, pages 225–235. ACM, 2007.

13 J. L. Massey. Guessing and entropy. In *Proc. of the 1994 IEEE International Symposium on Information Theory*, page 204, June 1994.

14 Joël Ouaknine. Decision problems for linear recurrence sequences. In L. Gasieniec and F. Wolter, editors, *FCT*, volume 8070 of *LNCS*, page 2. Springer, 2013.

15 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In Chandra Chekuri, editor, *SODA*, pages 366–379. SIAM, 2014.

16 Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *LNCS*, pages 288–302. Springer, 2009.

17 Hirotoshi Yasuoka and Tachio Terauchi. On bounding problems of quantitative information flow. *Journal of Computer Security*, 19(6):1029–1082, 2011.

Multiple-Environment Markov Decision Processes*

Jean-François Raskin and Ocan Sankur

Département d'Informatique, Université Libre de Bruxelles (U.L.B.), Belgium

Abstract

We introduce Multi-Environment Markov Decision Processes (MEMDPs) which are MDPs with a *set* of probabilistic transition functions. The goal in an MEMDP is to synthesize a single controller strategy with guaranteed performances against *all* environments even though the environment is unknown a priori. While MEMDPs can be seen as a special class of partially observable MDPs, we show that several verification problems that are undecidable for partially observable MDPs, are decidable for MEMDPs and sometimes have even efficient solutions.

1998 ACM Subject Classification D.2.4 Software/Program verification, F.3.1 Specifying and Verifying and Reasoning about Programs, G.3 Probabilities and Statistics

Keywords and phrases Markov decision processes, probabilistic systems, multiple objectives

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.531

1 Introduction

Markov decision processes (MDP) are a standard formalism for modeling systems that exhibit both stochastic and non-deterministic aspects. At each round of the execution of an MDP, an action is chosen by a controller (resolving non-determinism), and the next state is determined stochastically by a probability distribution associated to the current state and the chosen action. A controller is thus a *strategy* (a.k.a. *policy*) that determines which action to choose at each round according to the history of the execution so far. Algorithms for finite state MDPs are known for a large variety of objectives including omega-regular objectives [5], PCTL objectives [1], or quantitative objectives [18].

Multiple-Environment MDP (MEMDP). In an MDP, the environment is *unique*, and this may not be realistic: we may want to design a control strategy that exhibits good performances under several hypotheses formalized by different models for the environment, and those environments may *not* be distinguishable or we may *not want* to distinguish them (e. g. because it is too costly to design several control strategies.) As an illustration, consider the design of guidelines for a medical treatment that needs to work adequately for two populations of patients modeled by different stochastic models, even if the patients cannot be diagnosed to be in one population or in the other. This can be modeled by an MDP with two different models for the responses of the patients to the sequence of actions taken during the cure. We want a therapy that possibly makes decisions by observing the reaction of the patient and that works well (say reaches a good state for the patient with high probability) no matter if the patient belongs to the first of the second population.

Facing two potentially indistinguishable environments can be easily modeled with a partially observable MDPs. Unfortunately, this model is particularly intractable [3] (e. g. quantitative reachability, safety, and parity objectives, and even qualitative parity objectives

* Supported by the ERC starting grant inVEST (FP7-279499).

are undecidable.) To remedy to this situation, we introduce *multiple-environment MDPs (MEMDP)* which are MDPs with a *set* of probabilistic transition functions, rather than a *single one*. The goal in a MEMDP is to synthesize a single controller with guaranteed performances against *all* environments even though the environment which is operating is unknown a priori (it may be discovered during interaction but not necessarily.) We show that problems that are undecidable for partially observable MDPs, are decidable for MEMDPs and sometimes have even efficient solutions.

Results. We study MEMDPs with three types of objectives: reachability, safety and parity objectives. For each of those objectives, we study both *qualitative* and *quantitative* threshold decision problems. In this paper, we concentrate on MEMDPs with two environments as the two-environment case exhibits all the conceptual difficulties of the general case and it will easy the presentation of our results. The generalisation of the results for the n-environment case is left for future works. We first show that winning strategies may need infinite memory as well as randomization, and we provide algorithms to solve the decision problems. As it is classical, we consider two variants for the qualitative threshold problems. The first variant, asks to determine the existence of a *single* strategy that wins the objective with probability one (almost surely winning) in all the environments of the MEMDP. The second variant asks to determine the existence of a family of *single* strategies such that for all $\epsilon > 0$, there is one strategy in the family that wins the objective with probability larger than $1 - \epsilon$ (limit sure winning) in all the environments of the MEMDP. For both almost sure winning and limit sure winning, and for all three types of objectives, we provide efficient polynomial time algorithmic solutions. Then we turn to the quantitative threshold problem that asks for the existence of a single strategy that wins the objective with a probability that exceeds a given rational threshold in all the environments. We show the problem to be NP-hard (already for two environments and acyclic MEMDPs), so classical quantitative analysis techniques based on LP cannot be applied easily. Instead, we show that finite memory strategies are sufficient to approach achievable thresholds and we reduce the existence of bounded memory strategies to solving quadratic equations, leading to solutions in polynomial space. Our solutions rely on several new concepts (double-end components, good end-component, revealing edges, etc.) that bring deep understanding of the problems. The proofs are omitted due to space constraints, but a long version is available in [19].

Related Work. Interval Markov chains are Markov chains in which transition probabilities are only known to belong to given intervals (see *e. g.* [13, 14, 4]). Similarly, Markov decision processes with uncertain transition matrices for finite-horizon and discounted cases were considered [17]. The latter work also mentions the *finite scenario case* in which the transition probabilities are chosen among a finite set, as in our case. However, a solution is given only for the case where these probabilities can *independently* change in each step. Independence is a *simplifying assumption* that provides pessimistic guarantees. This means one ignores all information one might obtain on the system along observed histories, and so the results tend to be overly pessimistic.

Our work is related to reinforcement learning, where the goal is to develop strategies which ensure good performance in unknown environments, by learning and optimizing simultaneously; see [12] for a survey. In particular, it is related to the multi-armed bandit problem where one is given a set of systems with unknown reward distributions, and the goal is to choose the best one while optimizing the overall cost incurred while learning. The problem of finding the optimal one (without optimizing) with high confidence was considered in [9, 15], and is related to our constructions inside *distinguishing double end-components* (see Section 5). However, our problems differ from this one as in multi-armed bandit problem models of the bandits are unknown while our environments are known but we do not know a priori which environment is playing.

Multiple reachability objectives in MDPs were considered in [7]: given an MDP and multiple targets T_i, thresholds α_i, decide if there is a strategy forcing each T_i with probability at least α_i. Multiple reachability in MDPs can be seen as a special case of the reachability problem in MEMDPs (consider multiple copies of the same transition relation and for each copy one target set T_i) but there is no easy reduction the other way around. Indeed, while for multi-reachability objectives in MDPs with absorbing target states, optimal memoryless strategies always exist [7], we show that for reachability objectives in MEMDPs with absorbing target states, we may need infinite memory to play optimally. The former problem can be solved in polynomial time using linear programming; but we show that the quantitative reachability problem for MEMDPs with *two* environments and absorbing target states is NP-hard; so no polynomial time reduction to multi-reachability in MDPs is possible unless P=NP. An extension to multiple quantitative objectives were considered in [10], where finite-memory strategies also suffice and the algorithm is uses linear programming.

2 Definitions

A finite *Markov decision process (MDP)* is a tuple $M = (S, A, \delta)$, where S is a finite set of *states*, A a finite set of *actions*, where $A(s)$ denotes the set of actions available from $s \in S$, and $\delta : S \times A \to \mathcal{D}(S)$ a partial function defined for each pair (s, a) such that $a \in A(s)$, and $\mathcal{D}(S)$ is the set of *probability distributions* on S. We define a *run* of M as a finite or infinite sequence $s_1 a_1 \dots a_{n-1} s_n \dots$ of states and actions such that $\delta(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 1$. Finite runs are also called *histories* and denoted $\mathcal{H}(M)$.

Sub-MDPs and end-components. For the following definitions, we fix an MDP $M = (S, A, \delta)$. A *sub-MDP M' of M* is an MDP (S', A', δ') with $S' \subseteq S$, $\emptyset \neq A'(s) \subseteq A(s)$ for all $s \in S'$, and $\mathsf{Supp}(\delta(s,a)) \subseteq S'$ for all $s \in S', a \in A'(s)$ (here $\mathsf{Supp}(\cdot)$ denotes the support), and $\delta' = \delta|_{S' \times A'}$. By an abuse of notation, we may omit δ', and refer to the sub-MDP by (S', A'). For any subset $S' \subseteq S$ for which there exists a sub-MDP (S', A', δ'), let us denote by $M|_{S'}$ the *sub-MDP of M induced by S'*, which is the sub-MDP with the largest set of actions. In other terms, the sub-MDP induced by S' contains *all* actions of S' whose supports are inside S'. An MDP is strongly connected if between any pair of states s, t, there is a run. An *end-component* of $M = (S, A, \delta)$ is a sub-MDP $M' = (S', A', \delta')$ that is strongly connected. It is known that the union of two end components with non-empty intersection is an end-component; one can thus define *maximal* end-components. We let $\mathsf{MEC}(M)$ denote the set of maximal end-components of M, computable in polynomial time [6]. An *absorbing state s* is such that for all $a \in A(s)$, $\delta(s, a, s) = 1$.

Strategies. A *strategy* σ is a function $(SA)^*S \to \mathcal{D}(A)$ such that for all $h \in (SA)^*S$ ending in s, we have $\mathsf{Supp}(\sigma(h)) \subseteq A(s)$. A strategy is *pure* if all histories are mapped to *Dirac distributions*. A strategy σ can be encoded by a *stochastic Moore machine*, $(\mathcal{M}, \sigma_a, \sigma_u, \alpha)$ where $\mathcal{M}$ is a finite or infinite set of memory elements, α the *initial distribution on* $\mathcal{M}$, σ_u the *memory update function* $\sigma_u : A \times S \times \mathcal{M} \to \mathcal{D}(\mathcal{M})$, and $\sigma_a : S \times \mathcal{M} \to \mathcal{D}(A)$ the *next action function* where $\mathsf{Supp}(\sigma(s, m)) \subseteq A(s)$ for any $s \in S$ and $m \in \mathcal{M}$. We say that σ is *finite-memory* if $|\mathcal{M}| < \infty$, and *K-memory strategy* if $|\mathcal{M}| = K$; it is memoryless if $K = 1$, thus only depends on the last state of the history. Otherwise a strategy is *infinite-memory.* We define such strategies as functions $s \mapsto \mathcal{D}(A(s))$ for $s \in S$. An MDP M, a strategy σ encoded by $(\mathcal{M}, \sigma_a, \sigma_u, \alpha)$, and a state s determine a finite Markov chain M_s^σ defined on the state space $S \times \mathcal{M}$ as follows. The initial distribution is such that for any $m \in \mathcal{M}$, state (s, m) has probability $\alpha(m)$, and 0 for other states. For any pair of states (s, m) and (s', m'), the probability of the transition $(s, m), a, (s', m')$ is equal to $\sigma_a(s, m)(a) \cdot \delta(s, a, s') \cdot \sigma_u(s, m, a)(m')$. A *run* of M_s^σ is a finite or infinite sequence of the form $(s_1, m_1), a_1, (s_2, m_2), a_2, \ldots$, where each $(s_i, m_i), a_i, (s_{i+1}, m_{i+1})$ is a transition with nonzero probability in M_s^σ, and $s_1 = s$. In this case, the run $s_1 a_1 s_2 a_2 \ldots$, obtained by projection to M, is said to be *compatible with* σ. When considering the probabilities of events in M_s^σ, we will often consider sets of runs of M. Thus, given $E \subseteq (SA)^*$, we denote by $\mathbb{P}_{M,s}^\sigma[E]$ the probability of the runs of M_s^σ whose projection to M is in E. For any strategy σ in a MDP M, and a sub-MDP $M' = (S', A', \delta')$, we say that σ is *compatible with* M' if for any $h \in (SA)^* S'$, $\mathsf{Supp}(\sigma(h)) \subseteq A'(\mathsf{last}(h))$, where $\mathsf{last}(h)$ is the last state of h.

Let $\mathsf{Inf}(w)$ denote the disjoint union of states and actions that occur infinitely often in the run w; Inf is thus seen as a random variable. By an abuse of notation, we say that $\mathsf{Inf}(w)$ is equal to a sub-MDP D whenever it contains exactly the states and actions of D. It was shown that for any MDP M, state s, strategy σ, $\mathbb{P}_{M,s}^\sigma[\mathsf{Inf} \text{ is an end-component}] = 1$ [6]. We call a subset of states T *transient* if under all strategies, and starting from any state, almost surely, T is visited finitely many times.

Objectives. Given a set T of states, we define a *safety objective w.r.t.* T, written $\mathsf{Safe}(T)$, as the set of runs that only visit T. A *reachability objective w.r.t.* T, written $\mathsf{Reach}(T)$, is the set of runs that visit T at least once. We also consider *parity objectives.* A *parity function* is defined on the set of states $p : S \to \{0, 1, \ldots, 2d\}$ for some nonnegative integer d. The set of *winning runs of* M *for* p is defined as $\mathcal{P}_p = \{w \in (SA)^\omega \mid \min\{p(s) \mid s \in \mathsf{Inf}(w)\} \in 2\mathbb{N}\}$. For any MDP M, state s, strategy σ, and objective Φ, we denote $\mathsf{Val}_\Phi^\sigma(M, s) = \mathbb{P}_{M,s}^\sigma[\Phi]$ and $\mathsf{Val}_\Phi^*(M, s) = \sup_\sigma \mathbb{P}_{M,s}^\sigma[\Phi]$. We say that objective Φ is *achieved surely* if for some σ, all runs of M from s compatible with σ satisfy Φ. Objective Φ is *achieved with probability* α in M from s if for some σ, $\mathsf{Val}_\Phi^\sigma(M, s) \geq \alpha$. If Φ is achieved with probability 1, we say that it is *achieved almost surely.* Objective Φ is *achieved limit-surely* if for any $\epsilon > 0$, there exists a strategy σ_ϵ which achieves Φ with probability $1 - \epsilon$. In MDPs, limit-sure achievability coincides with almost-sure achievability since optimal strategies exist. We define $\mathsf{AS}(M, \Phi)$ as the set of states of M where Φ is achieved almost surely. Recall that for reachability, safety, and parity objectives these states can be computed in polynomial time, and are only dependent on the supports of the probability distributions [1, 6]. It is known that for any MDP M, state s, and a reachability, safety, or parity objective, there exists a pure memoryless strategy σ computable in polynomial time achieving the optimal value [18, 5].

In the next lemma, we recall that the optimal value inside any end-component is either 0 or 1, and that this only depends on the supports of the probability distributions.

▶ **Lemma 1** ([6]). *Let $M = (S, A, \delta)$ be a strongly connected MDP, and p a parity function. Then, for any MDP $M' = (S, A, \delta')$ such that for all $s \in S$, $a \in A$, $\mathsf{Supp}(\delta(s, a)) = \mathsf{Supp}(\delta'(s, a))$, and for all states $s \in S$, there exists a strategy σ such that $\mathsf{Val}_{\mathcal{P}_p}^\sigma(M, s) = \mathsf{Val}_{\mathcal{P}_p}^*(M, s) = \mathsf{Val}_{\mathcal{P}_p}^*(M', s) = \mathsf{Val}_{\mathcal{P}_p}^\sigma(M', s) \in \{0, 1\}$.*

3 Multiple-Environment MDP

A *multiple-environment MDP (MEMDP)*, is a tuple $M = (S, A, (\delta_i)_{1 \leq i \leq k})$, where for each i, (S, A, δ_i) is an MDP. We will denote by M_i the MDP obtained by fixing the edge probabilities δ_i, so that $\mathbb{P}^{\sigma}_{M_i,s}[E]$ denotes the probability of event E in M_i from state s under strategy σ. Intuitively, each M_i corresponds to the behavior of the system at hand under a different *environment*; in fact, while the state space is identical in each M_i, the transition probabilities between states and even their supports may differ. In this paper, we concentrate on the case of $k = 2$. We are interested in synthesizing a *single* strategy σ with guarantees on *both* environments, without a priori knowing against which environment σ is playing. We consider reachability, safety, and parity objectives, and again for readability, we consider the case where the same objective is to hold in all environments. The general quantitative problem is the following.

▶ **Definition 2.** Given MEMDP M, state s_0, $\alpha_1, \alpha_2 \in [0, 1]$, and Φ, a reachability, safety, or a parity objective, decide if there is a strategy σ such that $\forall i \in \{1, 2\}, \mathsf{Val}^{\sigma}_{\Phi}(M_i, s) \geq \alpha_i$.

We refer to the general problem as *quantitative reachability (resp. safety, parity)*. Given M, s_0, (α_1, α_2), Φ, we say that Φ is *achieved with probabilities* (α_1, α_2) in M from s if there is a strategy σ witnessing the above definition. We say that Φ is *achieved almost surely* in M from s if it is achieved with probabilities $(1, 1)$. Objective Φ is *achieved limit-surely* in M from s if for any $\epsilon > 0$, Φ is achieved in M from s with probabilities $(1-\epsilon, 1-\epsilon)$. *Almost-sure reachability (resp. safety, parity)* problems consist in deciding whether in a given M, from a state s, a given objective is achieved almost surely. *Limit-sure reachability (resp. safety, parity)* problems are defined respectively.

Given any MEMDP $M = (S, A, \delta_1, \delta_2)$, we define the MDP $\cup M = (S, A, \delta)$ by taking, for each action, the union of all transitions, and assigning them uniform probabilities. For any sub-MDP (S', A', δ') of $\cup M$, we define the *sub-MEMDP induced by the sub-MDP* (S', A', δ') as the MEMDP $(S', A', \delta'_1, \delta'_2)$ where $\delta'_i = \delta_i|_{S' \times A'}$. For any subset $S' \subseteq S$, the *sub-MEMDP of M induced by S'* is the sub-MEMDP of M induced by the sub-MDP of $\cup M$ induced by S'.

Strategy Complexity. Unlike MDPs, all considered objectives may require infinite memory and randomization, and Pareto-optimal probability vectors may not be achievable (a Pareto-optimal vector is component-wise maximal). All counterexamples are given in Fig. 1.

▶ **Lemma 3.** *For some MEMDPs M and objectives Φ:*

- *There exists a randomized strategy that achieves Φ with higher probabilities in both environments than any pure strategy,*
- *There exists an infinite-memory strategy that achieves Φ with higher probabilities in both environments than any finite-memory strategy,*
- *Objective Φ can be achieved limit-surely but not almost surely (showing Pareto-optimal vectors are not always achievable).*

Results. We give efficient algorithms for almost-sure and limit-sure cases:

(A) The almost-sure reachability, safety, and parity problems are decidable in polynomial time (Theorems 5 and 19). Finite-memory strategies suffice.

(B) The limit-sure reachability, safety, and parity problems are decidable in polynomial time (Theorems 12 and 20). Moreover, for any $\epsilon > 0$, to achieve probabilities of at least $1-\epsilon$, $O(\frac{1}{\eta^2}\log(\frac{1}{\epsilon}))$-memory strategies suffice, where η denotes the smallest positive difference between the probabilities of M_1 and M_2.

The general quantitative problem is harder as shown by the next results. We call an MEMDP *acyclic* if the only cycles are self-loops in all environments.

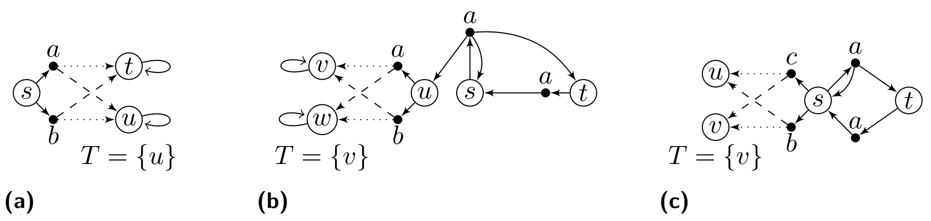


Figure 1 We adopt the following notation in all examples: edges that only exist in M_1 are drawn in dashed lines, and those that only exist in M_2 by dotted ones, and all probabilities are uniform unless otherwise said. To see that randomization may be necessary, observe that in the MEMDP M in Fig. 1a, the vector $(0.5, 0.5)$ of reachability probabilities for target T can only be achieved by a strategy that randomizes between a and b. In the MEMDP in Fig. 1b, where action a from s has the same support in M_1 and M_2 but different distributions. Any strategy almost surely reaches u in both M_i, since action a from s has nonzero probability of leading to u. Intuitively, the best strategy is to sample the distribution of action a from s, and to choose, upon arrival to u, either a or b according to the most probable environment. We prove that such an infinite-memory strategy achieves a Pareto-optimal vector which cannot be achieved by any finite-memory strategy. Last, in Fig. 1c, the MEMDP is similar to that of Fig. 1b except that action a from s only leads to s or t. We will prove in Section 6, that for any $\epsilon > 0$, there exists a strategy ensuring reaching T with probability $1 - \epsilon$ in each M_i. The strategy consists in sampling the distribution of action a from s a sufficient number of times and guessing the actual environment against which the controller is playing. However, the vector $(1, 1)$ is not achievable, which follows from Section 4.

(C) The quantitative reachability and safety problems are NP-hard on acyclic MEMDPs both for arbitrary and memoryless strategies (Theorem 13).

We can nevertheless provide procedures to solve the quantitative reachability and safety problems by fixing the memory size of the strategies.

(D) For any $K \geq 0$, the quantitative reachability and safety problems restricted to K-memory strategies can be solved in space polynomial in K and the size of M. (Theorem 14).

The quantitative parity problem can be reduced to quantitative reachability, so the previous result can also be applied for the quantitative parity problem.

(E) The quantitative parity problem can be reduced to quantitative reachability in polynomial time (Theorem 20).

We show that finite-memory strategies are not restrictive if we are interested in approximately ensuring given probabilities.

(F) Finite-memory strategies suffice to approximate quantitative reachability, safety, and parity problems up to any desired precision (Theorem 15).

We provide an *approximate* solution for quantitative reachability in the following sense. We consider *ϵ-gap problems* where the goal is to give a correct answer on negative instances that are "far" from the positive instances by ϵ, and on positive instances that are far from the negative instances by ϵ, while giving no guarantees in the rest of the input [8, 11].

▶ **Definition 4.** The *ϵ-gap problem for reachability* consists, given MEMDP M, state s, target set T, and probabilities α_1, α_2, in answering i YES if $\exists\sigma, \forall i = 1, 2, \mathbb{P}^{\sigma}_{M_i,s}[\mathsf{Reach}(T)] \geq \alpha_i$, ii NO if $\forall\sigma, \exists i = 1, 2, \mathbb{P}^{\sigma}_{M_i,s}[\mathsf{Reach}(T)] < \alpha_i - \epsilon$, iii and arbitrarily otherwise.

(G) There is a procedure for the ϵ-gap problem for quantitative reachability in MEMDPs (Theorem 16). The ϵ-gap problem is NP-hard (Theorem 17).

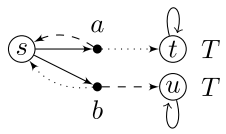


Figure 2 MEMDP M where $\mathsf{Reach}(T)$ can be achieved almost surely. In fact, $\mathsf{AS}(M_i, T) = \{s, t, u\}$ for all $i = 1, 2$, so $M' = M$, and $\mathsf{Val}_{\mathsf{Reach}(T)}(M'_i, s) = 1$ for $i = 1, 2$. The strategy returned by the algorithm consists in choosing, at s, a and b uniformly at random. Notice that there is no pure memoryless strategy achieving the objective almost surely.

Preprocessing. In an MEMDP with two environments, if one observes an edge that only exists in one environment, then the environment is known with certainty and any good strategy should immediately switch to the optimal strategy for the revealed environment. Formally, an edge (s, a, s') is i-*revealing* if $\delta_i(s, a, s') \neq 0$ and $\delta_{3-i}(s, a, s') = 0$. We make the following assumption w.l.o.g.:

▶ Assumption 1 (Revealed form). All MEMDPs $M = (S, A, \delta_1, \delta_2)$ are assumed to be in *revealed form*, that is, there exists a partition $S = S_u \uplus R_1 \uplus R_2$ satisfying the following properties. **1.** All states of R_1 and R_2 are absorbing in both environments, **2.** For any $i = 1, 2$, and any i-revealing edge (s, a, s'), we have $s' \in R_i$. Conversely, any edge (s, a, s') with $s' \in R_i$ is i-revealing. States R_i are called i-*revealed*, denoted $R_i(M)$. We write $R = R_1 \cup R_2$. The remaining states are called *unrevealed*.

In other words, we assume that any i-revealing edge leads to a known set of i-revealed states which are all absorbing. Assumption 1 can be made without loss of generality by redirecting any revealing edge to fresh absorbing states.

For any reachability (resp. safety) objective T, once a state in T (resp. $S \setminus T$) is visited the strategy afterwards is not significant since the objective has already been fulfilled (resp. violated). Thus, we assume that the set of target and unsafe states are absorbing.

▶ Assumption 2. For all considered objectives $\mathsf{Reach}(T)$ (resp. $\mathsf{Safe}(T')$), we assume that all target states T (resp. unsafe states $S \setminus T'$) absorbing for both environments.

Under assumptions 1 and 2, for any MEMDP M, and objective Φ, we denote $R_i^\Phi(M)$ the set of i-revealed states from which Φ holds almost surely in M_i, and define $R^\Phi(M) = R_1^\Phi(M) \cup R_2^\Phi(M)$. We will apply Assumption 1 throughout the paper, and Assumption 2 for reachability or safety objectives.

4 Almost-Sure Reachability

The algorithm for almost sure reachability is described in Algorithm 1. First, the state space is restricted to U since any state from which the objective holds almost surely in the MEMDP M must also belong to an almost surely winning state of each M_i. Second, we consider MEMDP M' induced by the states surely satisfying $\mathsf{Safe}(U)$ in $\cup M$. The problem is then reduced to finding an almost surely winning strategy in each M'_i separately. If such strategies exist, then we obtain our strategy by either 1) alternating between the two strategies using memory, or 2) randomizing between them.

Figure 2 is an example where almost-sure reachability holds; and we saw the example of Fig. 1c where almost-sure reachability does not hold.

▶ **Theorem 5.** *For any MEMDP M, objective $\mathsf{Reach}(T)$, and a state s, Algorithm 1 decides in polynomial time if $\mathsf{Reach}(T)$ can be achieved almost surely from s in M, and returns a witnessing memoryless randomized strategy.*

```
Input: MEMDP $M$, $\mathsf{Reach}(T)$, $s_0 \in S$
$U := \mathsf{AS}(M_1, \mathsf{Reach}(T)) \cap \mathsf{AS}(M_2, \mathsf{Reach}(T))$;
$M' :=$ Sub-MEMDP of $M$ induced by states $s$ s.t. $\mathsf{Val}^*_{\mathsf{Safe}(U)}(\cup M, s) = 1$;
if $\forall i = 1, 2$, $\mathit{Val}^*_{\mathit{Reach}(T)}(M'_i, s_0) = 1$ then
    Let $\sigma_i$ for $i = 1, 2$, such that $\mathsf{Val}^{\sigma}_{\mathsf{Reach}(T)}(M'_i, t) = 1$ for all $t \in U$;
    Return $\sigma'$ defined as $\sigma'(t) = \frac{1}{2}\sigma_1(t) + \frac{1}{2}\sigma_2(t)$, $\forall t \in S$;
else
    Return NO;
end
```

Algorithm 1. Almost-sure reachability algorithm for MEMDPs.

5 Double end-components

End-components play an important role in the analysis of MDPs; see *e. g.* [6]. Because the probability distributions in different environments of an MEMDP can have different supports, we need to adapt the notion for MEMDPs. We thus introduce *double end-components* which are sub-MDPs that are end-components in both environments.

Formally, given an MEMDP $M = (S, A, \delta_1, \delta_2, r)$, a *double end-component (DEC)* is a pair (S', A') where $S' \subseteq S$, and $A' \subseteq A$ such that (S', A') is an end-component in each M_i. A double end-component (S', A') is *distinguishing* if there is $(s, a) \in S' \times A'$ such that $\delta_1(s, a) \neq \delta_2(s, a)$. As the union of two DECs with a common state is a DEC, we consider *maximal DECs (MDEC)*. MDECs of M can be computed in polynomial time by eliminating from M all actions with different supports, then computing the MECs of $\cup M$. A DEC is *trivial* if it is an absorbing state.

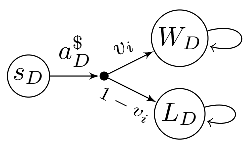
By Assumption 1 a DEC does not contain any revealed states unless it is trivial; therefore the supports of all DECs in both environments are identical. By Assumption 2, for reachability (resp. safety) objectives, non-trivial DECs do not contain target (resp. unsafe) states neither. Trivial DECs made of target (resp. safe) states are called *winning*. A DEC D is *winning* for a parity objective $\mathcal{P}_p$, if there is a strategy compatible with D satisfying $\mathcal{P}_p$ almost surely in both environments (a common strategy exists by Lemma 1).

Distinguishing DECs allow the strategy to learn the actual environment by sampling the distribution of distinguishing actions. One can in fact construct a strategy that surely stays inside a given DEC and guesses the actual environment with high confidence. Since a distinguishing DEC is non-trivial, the learning phase will surely avoid unsafe states. One then switches to the optimal strategy for the guessed environment:

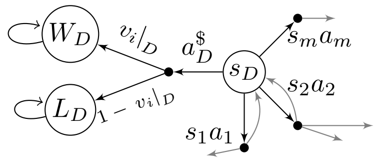
▶ **Lemma 6.** *Consider any MEMDP $M = (S, s_0, A, \delta_1, \delta_2)$, a distinguishing double end-component $D = (S', A')$, state $s \in S'$, $\epsilon > 0$, and any objective Φ reachability, safety, parity. For any $\epsilon > 0$, there exists a strategy σ such that $\mathbb{P}^{\sigma}_{M_i,s}[\Phi] \geq (1 - \epsilon)\mathit{Val}^*_{\Phi}(M_i, s), \forall i = 1, 2$.*

We now present a transformation for general MEMDPs by contracting DECs, which preserves the values up to any desired ϵ by Lemma 6. Given a DEC $D = (S', A')$, a *frontier state* s of D is such that there exists an action $a \in A(s) \setminus A'(s)$, which is not in D, index $i \in \{1, 2\}$, and $s' \notin S'$ such that $\delta_i(s, a, s') \neq 0$. An action $a \in A(s) \setminus A'(s)$ is a *frontier action* for D. A pair (s, a) is called *frontier state-action* when $a \in A(s)$ is a frontier action.

▶ **Definition 7.** Given an MEMDP $M = (S, A, \delta_1, \delta_2)$, and reachability or safety objective Φ, we define $\hat{M} = (\hat{S}, \hat{A}, \hat{\delta_1}, \hat{\delta_2})$ as follows. a) Any distinguishing MDEC D is contracted as in Fig. 3a where in M_i, action a leads to W_D with probability $v_i = \mathsf{Val}^*_{\Phi}(M_i, D)$, and to



(a) Reducing distinguishing DECs, where $v_i = \mathsf{Val}^*_{\phi_i}(M_i, D)$.



(b) Reduction of double end-components where $v_i|_D = \mathsf{Val}^*_\Phi(M_i|_D, D)$.

Figure 3 Reduction of double end-components.

L_D with probability $1 - v_i$. b) Any non-distinguishing MDEC $D = (S', A')$ is replaced with the module in Fig. 3b. The actions $a_D^\$$ and $\{f_i a_i\}_{(f_i,a_i)\in F}$ are available from s_D where F is the set of pairs of frontier state-actions of D. For any (f_i, a_i), the distribution $\hat{\delta}_j(s_D, f_i a_i)$ is obtained from $\delta_j(f_i, a_i)$ by redirecting to s_D all edges that lead inside S'. Define the new objective $\hat{\Phi}$ by restricting Φ to $\hat{S}$, and adding all states W_D in the target (resp. safe) set. We write $\hat{\mathcal{A}} : S \to \hat{S}$ (also denoted $s \mapsto \hat{s}$) the mapping from the states of S to those of $\hat{S}$.

The intuition is that when the play enters a *distinguishing* DEC D, by Lemma 6, we can arbitrarily approximate probabilities $v_i = \mathsf{Val}^*_\Phi(M_i, D)$; this is represented by action $a_D^\$$ in Fig. 3a. From a state s in a *non-distinguishing* DEC D in M, the play either stays forever inside and obtains the value $\mathsf{Val}^*_\Phi(M_1|_D, s) = \mathsf{Val}^*_\Phi(M_2|_D, s)$ (as it is non-distinguishing) – represented by $a_D^\$$ in Fig. 3b, or it eventually leaves D. The latter case is represented by the actions leading to frontier states, since D is necessarily left from such a state. Note that there is a strategy under which, from any state of D, in M_1 and M_2, all states of D, and in particular its frontier states, are visited infinitely often (by considering a memoryless strategy choosing all actions uniformly at random – see *e. g.* [18]). The equivalence between M and $\hat{M}$ for reachability and safety is shown next. Note that the value vectors are preserved although vectors achieved in $\hat{M}$ may not be achievable in M (see Fig. 1c).

▶ **Lemma 8.** *For MEMDPs M, and reachability of safety objectives Φ, $\mathsf{Val}^*_\Phi(M, s) = \mathsf{Val}^*_{\hat{\Phi}}(\hat{M}, \hat{s})$. Any end-component D of $\hat{M}_i$ is either a trivial DEC, or transient in $\hat{M}_{3-i}$.*

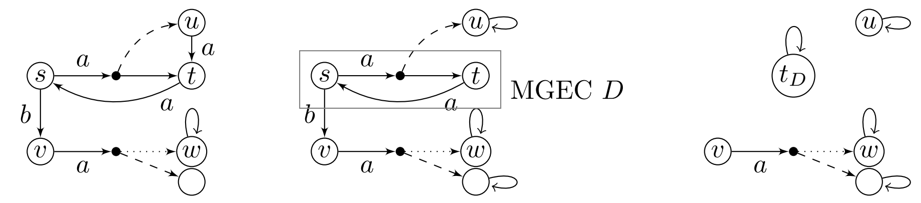
By Definition 7, and Lemmas 8-8, the following assumption can be made w.l.o.g.

▶ Assumption 3. All MEMDPs are assumed to have only trivial DECs.

6 Limit-Sure Reachability

In this section, we describe our polynomial-time algorithm for limit-sure reachability in MEMDPs. Throughout this section, we make Assumptions 1, 2, and 3.

We saw in the previous section how the strategy can safely learn the current environment with high confidence inside DECs. It turns out that it is possible to apply such learning strategies outside DECs. We need to introduce a new concept, called *good end-components* in order to fully capture all subsets of states where such a learning strategy can be applied. Consider the example of Fig. 4. Here, the MDP M_1 has a MEC D with the following property: the strategy σ compatible with D and choosing all actions of D uniformly at random, achieves the objective almost surely in M_2. In fact, a strategy that always chooses a at states s and t almost surely reaches u in M_2. In order to achieve the objective with probability close to 1, one can run strategy σ for a large number of steps, and if state u is still not reached, switch to the optimal strategy for M_1, that is, choose b from s. It can



(a) MEMDP M with $T = \{u, w\}$.

(b) Assumptions 1 and 2 satisfied. We have $T = \{u, w\}$.

(c) MEMDP $\widetilde{M}$ with $\widetilde{T} = \{t_D, u, w\}$ and $\widetilde{\mathcal{A}}(s) = t_D$.

Figure 4 On the left, an MEMDP with objective $\mathsf{Reach}(T)$, which is *not* in revealed form; an equivalent instance M in revealed form is shown in the middle. Note that M has only trivial DECs. States $\{s, t\}$ induce a *good end-component* D in M_2; the strategy choosing action a at s and t is almost surely winning in M_1. The construction $\widetilde{M}$ is shown on the right, where all states of D are contracted as t_D which is a target state. Because $\widetilde{\mathcal{A}}(s) = t_D$, Φ is achieved limit-surely from s.

be shown that such a strategy achieves the objective with probabilities $(1-\epsilon, 1-\epsilon)$, for any $\epsilon > 0$, from any state of such end-components. Here D is a *good end-component* of M_1.

Formally, an end-component D of M_i is *good* if the strategy that chooses all edges of D uniformly at random is almost sure winning for M_{3-i} from any state in D. Under any such strategy, any edge leaving D is revealing for M_{3-i}. Observe that the union of good end-components with non-empty intersection is a good end-component. We thus consider *maximal good end components (MGECs)* which can be computed in polynomial time.

▶ **Lemma 9.** *For any M, the MGECs of M_1 and M_2 can be computed in polynomial time.*

We define a transformation to MEMDPs by contracting MGECs since we know that one can learn the current environment from these states, without risking to lose.

▶ **Definition 10** (Transformation $\widetilde{M}$)**.** For any MEMDP M, and reachability objective Φ, we let $\widetilde{M} = (\widetilde{S}, \widetilde{A}, \widetilde{\delta}_1, \widetilde{\delta}_2)$ by applying the following transformation to M and Φ. Mark any state s that belongs to some MGEC D of M_i for some $i = 1, 2$, by D. If a state can be marked twice, choose one marking arbitrarily. We define $\widetilde{M}$ by redirecting any edge entering a state marked by some D to a fresh absorbing state t_D. For each $i = 1, 2$, the reachability objective $\widetilde{\Phi}$ is defined by the union of Φ, with all states t_D such that Φ can be ensured almost surely from D in M_i. We let $\widetilde{\mathcal{A}}(\cdot)$ be the mapping from the states M to those of $\widetilde{M}$.

Intuitively, DECs and MGECs cover all subsets of states in which one can learn the actual environment with high confidence; while in the absence of such components, limit-sure becomes equivalent to almost-sure. The following lemma establishes this property.

▶ **Lemma 11.** *For any MEMDP M, reachability objective Φ, and state s, Φ can be achieved limit surely in M from s if, and only if $\widetilde{\Phi}$ can be achieved almost surely in $\widetilde{M}$ from $\widetilde{\mathcal{A}}(s)$. Moreover, given an almost sure winning strategy for $\widetilde{M}$, for any $\epsilon > 0$, one can compute a strategy with memory $O(\frac{\log(\epsilon)}{\log(1-p)})$ for M, where p is the smallest nonzero probability, that achieves probabilities $1-\epsilon$, and this strategy can be computed.*

The steps of the limit-sure reachability algorithm are thus as follows: **1.** Contract DECs by Def. 7. **2.** Contract MGECs by Def. 10. **3.** Solve almost-sure reachability by Algorithm 1.

▶ **Theorem 12.** *The limit-sure reachability problem is decidable in polynomial-time.*

7 Quantitative Reachability

In this section, we study the quantitative reachability problem for MEMDPs. We first establish NP-hardness, suggesting that it is unlikely to have a polynomial-time algorithm, and that techniques based on linear programming often used for the quantitative analysis of MDPs (e. g. [18]) cannot be applied. We prove the hardness result by reduction from the product-partition problem [16].

▶ **Theorem 13.** *Given an MEMDP M, target set T, and $\alpha_1, \alpha_2 \in [0,1]$, it is NP-hard to decide whether for some strategy σ, $\mathbb{P}^{\sigma}_{M_i,s_0}[\mathsf{Reach}(T)] \geq \alpha_i$ for each $i = 1, 2$.*

As an upper bound on the above problem, we show that quantitative reachability for strategies with a fixed memory size can be solved in polynomial space. The algorithm consists in encoding the strategy and the probabilities achieved by each state and each environment, as a bilinear equation, and solving these in polynomial space in the equation size (see [2] for general polynomial equations).

▶ **Theorem 14.** *The quantitative reachability and safety problems for K-memory strategies can be solved in polynomial space in K and in the size of M.*

We now show that considering finite-memory strategies are hardly restrictive, in the sense that they can always be used to approximately achieve the values. We give a bound on strategy memories that is sufficient to approximate the value by given ϵ. The idea underlying the proof of the following theorem is that along a long execution in MEMDPs, with high probability, either one enters a subset of states that is identical in both environments, or one has gathered enough samples on probability distributions to guess the actual environment with high confidence.

▶ **Theorem 15.** *For any MEMDP M satisfying Assumption 3, reachability objective Φ, strategy σ, and $\epsilon > 0$, there exists a N-memory strategy σ' with $\forall i = 1, 2, \mathbb{P}^{\sigma'}_{M_i,s}[\Phi] \geq \mathbb{P}^{\sigma}_{M_i,s}[\Phi] - \epsilon$, where $N = (|S| + |A|)^{\frac{4|S|^3|A|^2}{p^{|S|}\eta^2}\log^3(1/\epsilon)}$, with p the smallest nonzero probability and $\eta = \min\{|\delta_1(s,a,s') - \delta_2(s,a,s')| \mid s, a, s' \text{ s.t. } \delta_1(s,a,s') \neq \delta_2(s,a,s')\}$.*

We derive our procedure by Theorems 14 and 15. The "gap" can be chosen arbitrarily small, and the procedure is used to distinguish instances that are clearly feasible from those that are clearly not feasible, while giving no guarantee in the borderline of size ϵ.

▶ **Theorem 16.** *There is a procedure that runs in $O(N \cdot |M|)$ space solving the ϵ-gap problem for quantitative reachability in MEMDPs.*

It turns out that even the ϵ-gap problem is NP-hard. We prove this by identifying instances where the achieved probabilities are *isolated*:

▶ **Theorem 17.** *The ϵ-gap problem for MEMDPs is NP-hard.*

8 Safety and Parity Objectives

▶ **Lemma 18.** *Limit-sure safety is equivalent to sure safety in MEMDPs, and can be decided in polynomial time.*

For quantitative safety, the results of the previous section can be adapted without difficulty.

We give a polynomial-time algorithm for almost sure parity objectives, consisting in **1.** restricting the states to almost surely winning ones for both M_i, **2.** solving almost sure reachability where all states that belong to winning end-components in M_1 or M_2 are targets.

▶ **Theorem 19.** *The almost-sure parity problem is decidable in polynomial time.*

We now describe a polynomial-time reduction from quantitative parity to quantitative reachability preserving value vectors. The idea is to allow the strategy to irreversibly switch to an optimal strategy for environment i from any MEC of M_i, and to represent this switch by a target absorbing state. Intuitively, the new reachability condition is equivalent to the parity objective for two reasons: first, all runs eventually enter an end-component and stay there, which roughly corresponds to this switch, and second, the transformation only adds new actions, so any strategy in the original MEMDP is still valid in the new one, and in particular *learning* strategies. It follows 1) a polynomial-time algorithm for the limit-sure parity problem, 2) any algorithm for quantitative reachability can be used to solve the quantitative parity problem. In particular, results of Section 7 applies to parity.

▶ **Theorem 20.** *The quantitative parity problem is polynomial-time reducible to the quantitative reachability problem. The limit-sure parity problem is solvable in polynomial time.*

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

2 John Canny. Some algebraic and geometric computations in pspace. In *STOC'88*, pages 460–467, New York, NY, USA, 1988. ACM.

3 Krishnendu Chatterjee, Martin Chmelik, and Mathieu Tracol. What is decidable about partially observable markov decision processes with omega-regular objectives. In *CSL*, volume 23 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

4 Taolue Chen, Tingting Han, and Marta Z. Kwiatkowska. On the complexity of model checking interval-valued discrete time markov chains. *Inf. Process. Lett.*, 113(7):210–216, 2013.

5 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, July 1995.

6 Luca de Alfaro. *Formal verification of probabilistic systems.* Ph.D. thesis, Stanford University, 1997.

7 Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science*, 4(4), 2008.

8 Shimon Even, Alan L. Selman, and Yacov Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

9 Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Pac bounds for multi-armed bandit and markov decision processes. In *COLT'02*, volume 2375 of *LNCS*, pages 255–270. Springer, 2002.

10 Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Quantitative multi-objective verification for probabilistic systems. In *TACAS'11*, volume 6605 of *LNCS*, pages 112–127. Springer, 2011.

11 Oded Goldreich. On promise problems (a survey in memory of Shimon Even [1935–2004]). *Manuscript*, 2005.

12 Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

13 Igor O. Kozine and Lev V. Utkin. Interval-valued finite markov chains. *Reliable computing*, 8(2):97–113, 2002.

14 Antonín Kučera and Oldřich Stražovský. On the controller synthesis for finite-state markov decision processes. In *FSTTCS 2005*, volume 3821 of *LNCS*, pages 541–552. Springer, 2005.

15 Shie Mannor and John N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *J. Mach. Learn. Res.*, 5:623–648, December 2004.

16 C. T. Ng, M. S. Barketau, T. C. E. Cheng, and Mikhail Y. Kovalyov. "Product Partition" and related problems of scheduling and systems reliability: Computational complexity and approximation. *European Journal of Operational Research*, 207(2):601–604, 2010.

17 Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

18 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

19 Jean-François Raskin and Ocan Sankur. Multiple-environment markov decision processes. *CoRR*, abs/1405.4733, 2014.

Summary-Based Inter-Procedural Analysis via Modular Trace Refinement

Franck Cassez[1], **Christian Müller**[2], **and Karla Burnett**[1]

1 **NICTA and UNSW, Australia**
2 **TU Munich, Germany**

Abstract

We propose a generalisation of trace refinement for the verification of inter-procedural programs. Our method is a top-down modular, summary-based approach, and analyses inter-procedural programs by building function summaries on-demand and improving the summaries each time a function is analysed. Our method is sound, and complete relative to the existence of a modular Hoare proof for a non-recursive program. We have implemented a prototype analyser that demonstrates the main features of our approach and yields promising results.

1998 ACM Subject Classification D.2.4 [Software Engineering] Software/Program Verification, D.3.1[Programming Languages] Formal Definitions and Theory, F.3.1 [Logics and Meanings of Programs] Specifying and Verifying and Reasoning about Program

Keywords and phrases Program verification, Hoare Logic, Refinement, Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.545

1 Introduction

Automated software verification has made tremendous progress in the last decade. Static analysis tools are routinely used to analyse source code and have revealed many subtle bugs.

We address the problem of designing a context-sensitive, scalable inter-procedural analysis framework. Our method is fully modular, and analyses each function without in-lining function calls but rather by using an input/output summary for the functions. This provides scalability. Context-sensitivity provides accuracy and is achieved by building function summaries in a top-down manner and being able to refine these summaries (on-demand) during the analysis. The result of our algorithm (when it terminates) is either a proof that a program is error-free or an inter-procedural counter-example that witnesses the error.

Our method is a modular inter-procedural extension of *refinement of trace abstraction* [13] and inherits the main features of this approach: it is sound and complete w.r.t. the existence of a modular Hoare proof for the non-recursive program and strictly more powerful than predicate abstraction refinement. Due to space limitation, technical proofs that were part of the submitted version are now omitted.

2 Example

We consider inter-procedural programs like P_1 in Listing 1. The variables in each function (or procedure) are either *input* (read-only), *output* or *local* (read-write) integer variables. The variable `m` is local to `main` and `n` is an output variable. The variables `p,q` are input variables of `inc` and `r` is the output variable. The semantics of a function call like `n = inc(1, m)` (line 3) is that the left-hand-side variables (`n`) are assigned the values of the corresponding output variables (`r`) at the end of the computation of the callee (`inc`).

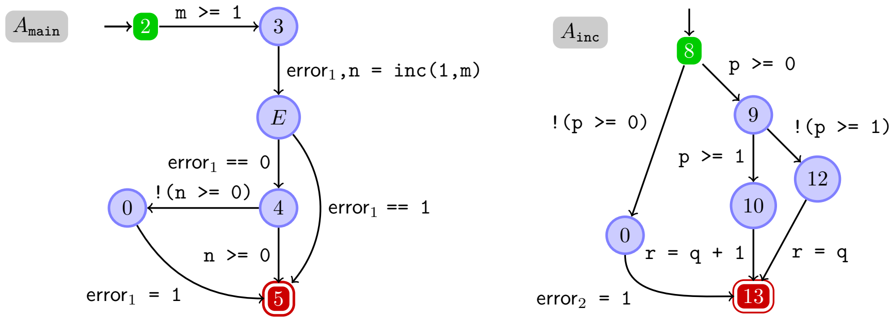


Figure 1 Function automata for program P_1.

Functions can be annotated with `assume` statements (e. g., `m >= 1` holds after line 2) and `assert` statements which assert predicates that should not be violated, otherwise the program contains an error. The objective is to check whether a program contains an execution that violates an assert statement. A function f is mapped to a function automaton, A_f, which is an extended *control flow graph (CFG)* with a single *entry* (green) and a single *exit* (red) node. To track `assert` statement violations across function boundaries in a modular manner, we introduce an extra[1] *output* variable, error, together with additional edges in the CFG of a function: *i*) a call like `n = inc(1,m)` is virtually expanded in `(error, n) = inc(1,m)` i. e., each function returns its internal error status; *ii*) each statement `assert(`φ`)` is viewed as an `if` statement, such that if φ does not hold the variable error is set to 1 and the control jumps to the exit node of the function; this is exemplified by edges 8 to 0 and 0 to 13 in automaton $A_{\mathtt{inc}}$; *iii*) as errors may occur during function calls, the status of the caller's error variable is checked after each function call: if it is set the control goes to the exit (red) node otherwise to the next instruction; this is exemplified by node E and edge E to 5 in automaton $A_{\mathtt{main}}$.

Program P_1 has two possible causes of error: one is to call `inc` with a negative value for parameter `p` (violating the assertion at line 8), and the other is to violate the assertion on `n` at line 4 in `main`. The (partial) correctness of `inc` can be expressed by the Hoare triple[2] $\{\neg\mathsf{error}_2\}$ `inc` $\{\neg\mathsf{error}_2\}$. To check whether the Hoare triple $\{\neg\mathsf{error}_2\}$ `inc` $\{\neg\mathsf{error}_2\}$ holds we try to find a counter-example trace: if we succeed we disprove the Hoare triple, if we fail the triple is valid. To do so, we view $A_{\mathtt{inc}}$ as a language acceptor with initial state *entry* (green) and final state *exit* (red) and the existence of a counter-example amounts to finding a *feasible* trace in $\mathcal{L}(A_{\mathtt{inc}})$, the language accepted by $A_{\mathtt{inc}}$. Determining whether such a feasible trace exists can be achieved using an iterative trace refinement algorithm [13, 15]: 1) Pick a trace[3] $w \in \mathcal{L}(A_{\mathtt{inc}})$; 2) Add the pre/post conditions $\neg\mathsf{error}_2/\mathsf{error}_2$ to the trace w and check whether

```
1   proc main() returns (n) {
2     assume(m >= 1);
3     n = inc(1, m);
4     assert(n >= 0);
5   }
6
7   proc inc(p,q) returns (r) {
8     assert(p >= 0);
9     if (p >= 1)
10        r = q + 1;
11    else
12        r = q;
13    endif;
14  }
```

Listing 1 Program P_1

[1] We introduce one indexed error_k variable per function to simplify the technical developments. We can equivalently use a global error variable to track the error status of a complete program.
[2] We use error (resp. $\neg$error) as a shorthand for "every valuation such that error is 1 (resp. 0)".
[3] Traces of length n are written $a_1 \cdot a_2 \cdots a_{n-1} \cdot a_n$.

the extended trace $w' = \neg\mathsf{error}_2 \cdot w \cdot \mathsf{error}_2$ is feasible; 3) If it is, we have found a counter-example and the triple is not valid. Otherwise, we look for a new trace in $\mathcal{L}(A_{\texttt{inc}}) \setminus \{w\}$. If $\mathcal{L}(A_{\texttt{inc}}) \setminus \{w\}$ is empty, the triple is valid; otherwise we start again at step 1 using a *refined* language $\mathcal{L}(A_{\texttt{inc}}) \setminus \{w\}$ which does not contain w. Notice that this process may not terminate[4]. A key result of [13] is that, for each infeasible trace w, a *rejecting automaton*, $A(w)$, can be computed that accepts traces that are infeasible for the same reasons as w. Thus in the refinement step 3), we can remove all the traces accepted by automaton $A(w)$ and not only $\{w\}$.

The outcome of the iterative trace refinement algorithm (when it terminates) is either a counter-example path or a confirmation that a triple holds. Our first result (Section 4) is that, when we establish that $\{P\}$ `f` $\{Q\}$ holds, we get a *better* triple $\{P'\}$ `f` $\{Q'\}$ with $P \implies P'$ (weaker assumption on input) and $Q' \implies Q$ (stronger constraint on input/output relation). Our main result (Section 5) is the extension of the trace refinement approach [13] to check whether triples hold for inter-procedural programs without in-lining function calls.

The main idea of our extension is illustrated next. A function call is viewed as a standard instruction: the call `r = f(m)` defines a relation between the input variables[5] `m` and the output variables `r`. The only difference to a standard instruction is that we do not exactly know this relation, which is the *strongest postcondition* operator for the function.

This can be remedied as follows: for each function call to `f`, we use a *summary* which is an over-approximation of the strongest postcondition of the function `f`. A summary for `inc` could be $\mathtt{p} \geq 1 \implies \mathtt{r} \leq \mathtt{q} + 1$ or, if we do not know anything about `inc`, $\mathit{True} \implies \mathit{True}$, which means that the output variables can be assigned any value. To determine the status of triple (H) $\{\neg\mathsf{error}_1\}$ `main` $\{\neg\mathsf{error}_1\}$, we try to find a witness trace in `main` invalidating it, i. e., starting in $\neg\mathsf{error}_1$ and ending in error_1:

1. let $w_1 =$ `m >= 1` $\cdot$ error_1`,n = inc(1,m)` $\cdot$ error_1 be a trace in $\mathcal{L}(A_{\texttt{main}})$.
2. Using the semantics of each statement, and the over-approximate summary semantics $(\mathit{True}, \mathit{True})$ for the function call, check whether $w'_1 = \neg\mathsf{error}_1 \cdot w_1 \cdot \mathsf{error}_1$ is feasible. It is and we get a witness *assignment* for the values of the variables in w'_1 that implies a pre/postcondition $\pi_1 : \mathtt{p} = 1 \wedge \mathtt{q} = 1 \wedge \neg\mathsf{error}_2/\mathsf{error}_2$ for `inc` to make w'_1 feasible.
3. To determine whether π_1 can be satisfied, we can establish the status of the *opposite* triple $\{\mathtt{p} = 1 \wedge \mathtt{q} = 1 \wedge \neg\mathsf{error}_2\}$ `inc` $\{\neg\mathsf{error}_2\}$.
   This triple *holds* and thus the corresponding witness pre/postcondition π_1 in `main` is infeasible. While establishing this, we have computed a *stronger valid* triple, $\{\mathtt{p} \geq 0 \wedge \neg\mathsf{error}_2\}$ `inc` $\{\neg\mathsf{error}_2\}$ that can, from now on, be used as a valid summary $(G_1, S_1) = (\mathtt{p} \geq 0 \wedge \neg\mathsf{error}_2, \neg\mathsf{error}_2)$ for `inc`.
4. We again check the feasibility of w'_1, this time with the new summary (G_1, S_1). Using (G_1, S_1), w'_1 becomes infeasible and thus w_1 can be ruled out.
5. We pick a different trace $w_2 =$ `m >= 1` $\cdot$ error_1`,n = inc(1,m)` $\cdot$ $\neg\mathsf{error}_1$ $\cdot$ `!(n >= 0)` $\cdot$ $\mathsf{error}_1 = 1$ and check whether the extended trace $w'_2 = \neg\mathsf{error}_1 \cdot w_2 \cdot \mathsf{error}_1$ is feasible. It is *provided* the call to `inc` can realise an input/output constraint given by a witness assignment for w'_2. Function `inc` cannot realise this witness assignment and the result of this check is a new valid triple for `inc`, $(G_2, S_2) = (\mathtt{p} \geq 1 \wedge \mathtt{q} \geq 1, \mathtt{r} \geq \mathtt{q} + 1)$.
6. We again check the feasibility of w'_2 with (G_1, S_1) and (G_2, S_2). w'_2 is now declared infeasible with (G_1, S_1) and (G_2, S_2) and this enables us to rule out w_2 in `main`.
7. There is only one trace left to explore in `main` but it cannot set error_1. The final result is a triple $\{\neg\mathsf{error}_1\}$ `main` $\{\neg\mathsf{error}_1 \wedge \mathtt{n} \geq \mathtt{m} + 1\}$ that is stronger than the initial one.

[4] Verifying C-like programs is undecidable.
[5] `m` and `r` are vectors of variables, however for clarity we omit vector notation.

3 Preliminaries

Programs are written in a simple inter-procedural programming language as commonly assumed [2, 18]. There are no pointers, no global variables, and we restrict to integer variables. This last restriction is not important as integer variables are expressive enough to encode a very large class of errors in imperative programs e. g., *array-out-of-bounds*, *NULL pointer dereferences*, etc. We assume a set of predicates over a set of variables e. g., in Quantifier-Free Linear Integer Arithmetic. Given a predicate φ, $\textsc{Var}(\varphi)$ is the set of variables appearing in φ. We freely use the logical notation or set notation depending on which is best suited, e. g., given two predicates P and Q, we use $P \wedge Q$ (logical and) or $P \cap Q$ (set intersection). $False$ corresponds to the empty set and $True$ to the set of all possible values.

The set of program *statements* Σ is comprised of: (i) *simple assignments* e. g., `y = t` where `y` is a variable and `t` a linear term on variables, (ii) *assume* statements which are predicates over the variables and (iii) *function calls* of the form `r1,···,rk = f(d1,···,dn)` where `f` is a function and `r1,···,rk` and `d1,···,dn` are the input and output variables.

Given a simple assignment st and a predicate φ, $\mathsf{post}(st, \varphi)$ is the *strongest condition* that holds after executing st from φ. For an assume statement st, the semantics are $\mathsf{post}(st, \varphi) = \varphi \wedge st$. The semantics of each function call are given by the strongest postcondition operator post for the function (although we may not explicitly have it). The post operator extends straightforwardly to *traces* in Σ^*.

A trace t satisfies a *pre/post condition* (P, Q) if $\mathsf{post}(t, P) \subseteq Q$. A trace t is (P, Q)-*feasible* if $\mathsf{post}(t, P) \cap Q \not\subseteq False$, otherwise it is *infeasible*. We let $\mathsf{Infeas}(P, Q)$ be the set of traces over Σ^* that are (P, Q)-infeasible. A trace t is *infeasible* if it is $(True, True)$-infeasible (or if it satisfies $(True, False)$), otherwise it is *feasible*; Infeas is the set of infeasible traces.

A *trace automaton* [13, 15] is a tuple $A = (\mathrm{L}, \delta, \mathrm{L}^{init}, \mathrm{L}^{exit})$ where L is a finite set of locations, $\delta \subseteq \mathrm{L} \times \Sigma \times \mathrm{L}$ is the transition relation, and $\mathrm{L}_{init}, \mathrm{L}_{exit} \subseteq \mathrm{L}$ are the initial and final (accepting) locations. The *language accepted by* A is $\mathcal{L}(A)$.

A function `f` is represented by a function automaton which is a trace automaton $A_{\mathtt{f}} = (\mathrm{L}_{\mathtt{f}}, \delta_{\mathtt{f}}, \{init_{\mathtt{f}}\}, \{exit_{\mathtt{f}}\})$. It is obtained from the CFG of `f` by adding the edges setting the error variable to encode `assert` statement violations (see $A_{\mathtt{main}}$ and $A_{\mathtt{inc}}$ in Figure 1).

4 Checking Intra-Procedural Partial Correctness

We assume in this section that functions do not contain function calls. We show how to construct automata that accept traces that satisfy Hoare triples. This extends the results of [13]. A similar development is accounted for in [15] but we establish here a new useful result in Theorem 5. Given a trace automaton A and two predicates P, Q (over the variables of A), the Hoare triple $\{P\}\ A\ \{Q\}$ is *valid* iff $\forall t \in \mathcal{L}(A), \mathsf{post}(t, P) \subseteq Q$. Program (or function) correctness [13] is defined by: $\{P\}$ `f` $\{Q\}$ is valid iff $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ is valid. The validity of a Hoare triple $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ can be expressed in terms of language inclusion:

▶ **Theorem 1.** *$\{P\}\ A_{\mathtt{f}}\ \{Q\}$ is valid iff $\mathcal{L}(A_{\mathtt{f}}) \subseteq \mathsf{Infeas}(P, \neg Q)$.*

We also extend the notion of inductive interpolants [13] for infeasible traces to (P, Q)-interpolants for (P, Q)-infeasible traces. Let $t = st_1 \cdots st_k$ be a (P, Q)-infeasible trace. A sequence of predicates $I_0, I_1, \cdots, I_k$ is a (P, Q)-*interpolant* for t if: 1) $P \implies I_0$, 2) $\forall 1 \leq i \leq k, \mathsf{post}(st_i, I_{i-1}) \subseteq I_i$ and 3) $I_k \wedge Q = False$. For $t \in \mathsf{Infeas}(P, Q)$, we let $\mathsf{itp}_{P,Q}(t)$ be the set of (P, Q)-interpolants for t. For $t \in \mathsf{Infeas}$, we let $\mathsf{itp}(t)$ be the set of interpolants for t. By Craig's interpolation theorem [10], we know that $\mathsf{itp}(t) \neq \varnothing$. It follows that:

▶ **Lemma 2.** *If* $t \in \mathsf{Infeas}(P,Q)$ *then* $\mathsf{itp}_{P,Q}(t) \neq \varnothing$.

Notice that the ability to compute actual interpolants (e. g., using SMT-solvers) is inessential as inductive interpolant can always be obtained using weakest preconditions for t.

Let $t = st_1 \ \cdots \ st_k$ be an infeasible trace and $\mathcal{I} = I_0, I_1, \cdots, I_k$ be an interpolant for t. The *canonical interpolant automaton* [13] for $(t, \mathcal{I})$ is a trace automaton $A^t_{\mathcal{I}} = (\mathrm{L}_{\mathcal{I}}, \delta_{\mathcal{I}}, \{init_{\mathcal{I}}\}, \{exit_{\mathcal{I}}\})$. An important property of canonical interpolant automata is that they accept sets of infeasible traces:

▶ **Theorem 3** ([13]). *If* $t \in \mathsf{Infeas}$ *and* $\mathcal{I} \in \mathsf{itp}(t)$ *then* $\mathcal{L}(A^t_{\mathcal{I}}) \subseteq \mathsf{Infeas}$.

We extend the definition of canonical interpolant automata to (P,Q)-*interpolant automata*. Let $t = st_1 \cdot st_2 \ \cdots \ st_k \in \mathsf{Infeas}(P,Q)$ and $\mathcal{I} = I_0, I_1, \cdots, I_k \in \mathsf{itp}_{P,Q}(t)$. Then t is also in $\mathsf{Infeas}(I_0, Q)$. Let $t' = \mathtt{assume}(I_0) \cdot t \cdot \mathtt{assume}(Q)$ (in the sequel we write such a trace $I_0 \cdot t \cdot Q$ omitting the $\mathtt{assume}$ statements). As $t \in \mathsf{Infeas}(I_0, Q)$, we have $t' \in \mathsf{Infeas}$ and moreover $\mathcal{I}' = True, I_0, \cdots, I_k, False$ is an interpolant for t'. We can then build the canonical interpolant automaton $A^{t'}_{\mathcal{I}'} = (L_{\mathcal{I}'}, \delta_{\mathcal{I}'}, \{init_{\mathcal{I}'}\}, \{exit_{\mathcal{I}'}\})$ for $(t', \mathcal{I}')$. We define the corresponding (P,Q)-interpolant automaton for $(t, \mathcal{I})$ as the tuple $A(P,Q)^t_{\mathcal{I}} = (\mathrm{L}_{\mathcal{I}}, \delta_{\mathcal{I}}, \mathrm{L}^{init}_{\mathcal{I}}, \mathrm{L}^{exit}_{\mathcal{I}})$ where: 1) $\mathrm{L}_{\mathcal{I}} = L_{\mathcal{I}'} \setminus \{init_{\mathcal{I}'}, exit_{\mathcal{I}'}\}$, 2) $\mathrm{L}^{init}_{\mathcal{I}} = \{\ell \in L_{\mathcal{I}'} \mid (init_{\mathcal{I}'}, P, \ell) \in \delta_{\mathcal{I}'}\}$ and $\mathrm{L}^{exit}_{\mathcal{I}} = \{\ell \in L_{\mathcal{I}'} \mid (\ell, Q, exit_{\mathcal{I}'}) \in \delta_{\mathcal{I}'}\}$ and 3) $\delta_{\mathcal{I}} = \delta_{\mathcal{I}'} \cap (\mathrm{L}_{\mathcal{I}} \times \Sigma \times \mathrm{L}_{\mathcal{I}})$. (P,Q)-interpolant automata accept sets of (P,Q)-infeasible traces:

▶ **Theorem 4.** *If* $t \in \mathsf{Infeas}(P,Q)$ *and* $\mathcal{I} \in \mathsf{itp}_{P,Q}(t)$ *then* $\mathcal{L}(A(P,Q)^t_{\mathcal{I}}) \subseteq \mathsf{Infeas}(P,Q)$.

We can now introduce Algorithm 1, *Hoare*$_1$, that can establish the status of a Hoare triple. If the triple is valid, it returns a new *stronger* triple and otherwise a witness counter-example. Algorithm 1 works as follows: the family of automata $A_i, i \geq 1$ accept only infeasible traces. If the condition of the while loop (line 1) is True, there is a trace t in the CFG of $A_{\mathtt{f}}$ that has not been declared $(P, \neg Q)$-infeasible yet. This $(P, \neg Q)$-infeasibility of trace t is investigated: if it is feasible, the triple $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ does not hold and t is a witness (line 4). If it is infeasible, an interpolant automaton A_n is built from t (lines 6 to 9) and added to the family $A_i, i \geq 1$. If the condition of the while loop (line 1) is False, all the traces of the CFG of $A_{\mathtt{f}}$ have been declared $(P, \neg Q)$-infeasible and hence $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ holds. Moreover, as stated by Theorem 5 the interpolants collected at each step during the refinement can be used to build a stronger triple (line 10).

A triple $\{P'\}\ \mathtt{f}\ \{Q'\}$ is *stronger* than $\{P\}\ \mathtt{f}\ \{Q\}$ if $P \implies P'$ and $Q' \implies Q$.

Line 9 of Algorithm 1 stores the interpolants each time a trace t is declared $(P, \neg Q)$-infeasible. It collects the interpolants I_0 and I_k and stores them in the arrays P_n and Q_n (each interpolant automaton A_n is also stored). If the triple is valid, the interpolant automata A_i *cover* the set of traces of $A_{\mathtt{f}}$, and $A_{\mathtt{f}}$ satisfies the triple $\{\cap^n_{i=1} P_i\}\ A_{\mathtt{f}}\ \{\cup^n_{i=1} Q_i\}$.

▶ **Theorem 5.** *If Algorithm 1 terminates and returns* $\mathsf{itp}(P', Q')$ *then* $\{P'\}\ A_{\mathtt{f}}\ \{Q'\}$ *is valid and stronger than* $\{P\}\ A_{\mathtt{f}}\ \{Q\}$.

If Algorithm 1 terminates it either returns: *i*) $\mathsf{path}(t)$, and then $\{P\}\ \mathtt{f}\ \{Q\}$ does not hold and t is such that $\mathsf{post}(t, P) \cap \neg Q \not\subseteq False$, or *ii*) $\mathsf{itp}(P', Q')$ then $\{P'\}\ \mathtt{f}\ \{Q'\}$ holds and $P \implies P'$ and $Q' \implies Q$. Claim *i*) holds because there are no function calls and the trace t selected at line 2 is such that $\mathsf{post}(t, P) \cap \neg Q \not\subseteq False$. As post is exact for statements which are not function calls, t is a counter-example. For *ii*), if *Hoare*$_1$ returns $\mathsf{itp}(P', Q')$, Theorem 5 holds proving correctness in this case. Termination of Algorithm 1 is of course not guaranteed as the verification problem for C-like programs is undecidable.

```
Algorithm 1: $Hoare_1(A_{\mathtt{f}}, P, Q)$
  Input  : A function automaton $A_{\mathtt{f}}$, two predicates $P$ and $Q$.
  Result :  $\mathsf{itp}(P', Q')$ with $P \implies P'$ and $Q' \implies Q$ if $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ holds,
            $\mathsf{path}(t)$ with $\mathsf{post}(P, t) \cap \neg Q \not\subseteq False$ otherwise.
  Var    : $\overline{A}$: arrays of interpolant automata, initially empty
           $\overline{P}, \overline{Q}$: arrays of predicates, initially empty
           $n$ : integer, initially 0
1  while $\mathcal{L}(A_{\mathtt{f}}) \not\subseteq \cup_{i=1}^{n} \mathcal{L}(A_i)$ do
       /* There is a trace $t$ from entry to exit in $\mathcal{L}(A_{\mathtt{f}}) \setminus \cup_{i=1}^{n} \mathcal{L}(A_i)$ */
2      Let $t = st_1 \cdots st_k \in \mathcal{L}(A_{\mathtt{f}}) \setminus \cup_{i=1}^{n} \mathcal{L}(A_i)$;
3      if $\mathsf{post}(t, P) \cap \neg Q \not\subseteq False$ then
           /* $\mathsf{post}(t, P) \subseteq Q$ does not hold, $t$ is a counter-example */
4          return $\mathsf{path}(t)$;
5      else
           /* $t$ is $(P, \neg Q)$-infeasible, we refine and iterate */
6          Let $\mathcal{I} = I_0, \cdots, I_k \in \mathsf{itp}_{P,\neg Q}(t)$;
7          Let $n := n + 1$;
8          Let $A_n = A(P, \neg Q)^{t}_{\mathcal{I}}$;
9          let $P_n := I_0$ and $Q_n := I_k$;
10 return $\mathsf{itp}(\cap_{i=1}^{n} P_i, \cup_{i=1}^{n} Q_i)$;
```

However, similar to the trace refinement algorithm of [13], we can ensure *incrementality* and *progress*. *Incrementality* is relative to the Hoare triple we are checking, and means that once a $(P, \neg Q)$-interpolant automaton has been computed it holds during the call to $Hoare_1(\mathtt{f}, P, Q)$ and we never have to withdraw it. *Progress* is ensured because if we discover a $(P, \neg Q)$-infeasible trace t at the n-th iteration of the while loop, it is accepted by the corresponding automaton A_n and thus cannot be found in subsequent rounds. As pointed out in [13], soundness of the algorithm i. e., if it terminates and declares a program error-free the program is actually error-free, is implied by Theorem 5. Completeness i. e., if a program is error-free, $Hoare_1$ can declare it error-free, holds (as usual) for a trivial reason [13] and we do not detail this.

5 Inter-Procedural Trace Refinement

Let `f` be a function with (formal) input parameters $\overline{x} = x_1, \ldots, x_k$ and output parameters $\overline{y} = y_1, \cdots, y_n$. We assume that input variables are not modified by a function and there are no global variables[6]. A *summary* $S(\overline{x}, \overline{y})$ for `f` is a set of pairs of predicates $(P_i(\overline{x}), Q_i(\overline{x}, \overline{y}))_{1 \leq i \leq n}$ where $\varphi(\overline{z})$ indicates that $\textsc{Var}(\varphi) = \{z_1, \cdots, z_n\}$. A summary can equivalently be viewed as a predicate $\wedge_{i=1}^{n} \big(P_i(\overline{x}) \implies Q_i(\overline{x}, \overline{y})\big)$.

The *exact* post operator is not explicitly available for functions. We want to over-approximate it while retaining enough information to prove a property of interest. We approximate the post operator for functions using summaries. A *context* $\mathcal{C}$ is a mapping from functions to summaries such that for each function `f` in the program, $\mathcal{C}(\mathtt{f})$ is a summary for

[6] These limitations are not compulsory but are commonly admitted [2, 18].

`f`. Given a context $\mathcal{C}$ we define an associated *summary post operator* $\widehat{\mathsf{post}}$ as follows:

$$\widehat{\mathsf{post}}(\mathcal{C}, st, \varphi) = \begin{cases} \mathsf{post}(st, \varphi) \text{ if } st \text{ is not a function call, or} \\ \exists \mathtt{r}.\varphi \wedge \mathcal{C}(\mathtt{f})(\mathtt{m}, \mathtt{r}) \text{ if } st \text{ is the function call } \mathtt{r = f(m)}. \end{cases}$$

In other words, only function call `post` operators are computed using summaries while other statements' strongest postcondition operators are preserved. As a function call `r = f(m)` only alters the output variables `r`, the *projection* of the predicate φ on the other variables, $\exists \mathtt{r}.\varphi$, remains true after the execution of `r = f(m)`. Moreover the target result variable `r` should satisfy the constraints between the formal input and output variables $\mathcal{C}(\mathtt{f})(\mathtt{m}, \mathtt{r})$ of `f`. $\mathcal{C}$ is an *over-approximating context* (in short over-approximation) if for every function automaton $A_{\mathtt{f}}$, every trace $t \in \mathcal{L}(A_{\mathtt{f}})$ and every predicate φ, $\mathsf{post}(t, \varphi) \subseteq \widehat{\mathsf{post}}(\mathcal{C}, t, \varphi)$. The definitions we introduced so far are also valid for the $\widehat{\mathsf{post}}$ operator: a trace t is (P, Q)-infeasible in context $\mathcal{C}$ if $\widehat{\mathsf{post}}(\mathcal{C}, t, P) \cap Q \subseteq \mathit{False}$, otherwise it is feasible (in this context). We write t is $\mathcal{C}$-(P, Q)-feasible (resp. infeasible) for t is (P, Q)-feasible (resp. infeasible) in context $\mathcal{C}$ If a context is an over-approximation, infeasibility in $\mathcal{C}$ implies infeasibility with the *exact* strongest postcondition `post` operators for the functions called. However, a trace may be (P, Q)-feasible in $\mathcal{C}$, but not (P, Q)-feasible with the *exact* `post` operator for each function. Valid Hoare triples can be used as over-approximations for functions:

▶ **Proposition 6.** *Let (P, Q) be two predicates on the input and input/output variables of* `f` *such that $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ holds. Then* $\mathsf{post}(\mathtt{r} = \mathtt{f}(\mathtt{m}), \varphi) \subseteq \widehat{\mathsf{post}}(\mathcal{C} : \mathtt{f} \mapsto \{(P, Q)\}, \mathtt{r} = \mathtt{f}(\mathtt{m}), \varphi)$.

Proposition 6 generalises to summaries that are sets of pairs of predicates:

▶ **Theorem 7.** *Let $\mathcal{C}$ be an over-approximation,* `f` *a function and (P, Q) be two predicates on the input and input/output variables of* `f` *such that $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ holds. The context $\mathcal{C}'$ defined by $\mathcal{C}'(h) = \mathcal{C}(h)$ for $h \neq \mathtt{f}$ and $\mathcal{C}'(\mathtt{f}) = \mathcal{C}(\mathtt{f}) \cup \{(P, Q)\}$ is an over-approximation.*

We can now propose our new modular trace refinement algorithm (Algorithms 2 and 3). The main difference between Algorithm 1 and Algorithm 2 is in how feasibility of a trace is checked using the summaries (call to *Status* at line 3). This step is more involved and is detailed in Algorithm 3. Algorithm 2 determines the status of a candidate triple $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ and either returns an *inter-procedural (counter-example) path* or a *stronger* triple $\{P'\}\ A_{\mathtt{f}}\ \{Q'\}$. An *inter-procedural path* for a trace t of function `g` is inductively defined by a mapping `path` such that:

- for statements st in t that are not function calls, $\mathsf{path}(st) = st$,
- for st a function call `r = f(m)`, $\mathsf{path}(st)$ is an inter-procedural path for `f`.

n Algorithm 2, we assume that the context variable $\mathcal{C}$ is a *global variable* and is initialised with default summaries e. g., $(\mathit{True}, \mathit{True})$ for each function. In Algorithm 2, line 3, the call to $\mathit{Status}(t, P, \neg Q)$ (*Status* is defined in Algorithm 3) returns the status of t: it is either $(P, \neg Q)$-feasible and $(\mathit{True}, \mathsf{path}(t))$ is returned or $\mathcal{C}$-$(P, \neg Q)$-infeasible and $(\mathit{False}, \bot)$ is returned ($\bot$ stands for the void path). Hoare_2 is very similar to Algorithm 1 once the status of a trace t is determined:

- if t is $\mathcal{C}$-$(P, \neg Q)$-infeasible i. e., $\mathit{Status}(t, P, \neg Q) = (\mathit{False}, \bot)$, then it is $(P, \neg Q)$-infeasible ($\mathcal{C}$ is an over-approximation) and we can compute an interpolant automaton to reject similar traces (lines 7 to 10 in Algorithm 2). Note also that the summary for the currently analysed function is updated (line 11).
- otherwise t is $(P, \neg Q)$-feasible i. e., $\mathit{Status}(t, P, \neg Q) = (\mathit{True}, \mathsf{path}(t))$ and t is an inter-procedural counter-example.

```
Algorithm 2: $Hoare_2(A_{\mathtt{f}}, P, Q)$
Global: $\mathcal{C}$: context with summaries for each function, initially $(True, True)$
Input : A function automaton $A_{\mathtt{f}}$, two predicates $P$ and $Q$.
Result : If $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ holds, $\mathsf{itp}(P', Q')$ with $P \implies P'$ and $Q' \implies Q$
         Otherwise an inter-procedural path $\mathsf{path}(t)$ with $\mathsf{post}(P, t) \cap \neg Q \not\subseteq False$.
Local : $\overline{A}$: arrays of interpolant automata, initially empty
        $\overline{P}, \overline{Q}$: arrays of predicates, initially empty
        $n$: integer, initially 0
/* Main refinement loop */
1  while $\mathcal{L}(A_{\mathtt{f}}) \not\subseteq \cup_{i=1}^{n} \mathcal{L}(A_i)$ do
       /* There is a trace from entry to exit in $\mathcal{L}(A_{\mathtt{f}}) \setminus \cup_{i=1}^{n} \mathcal{L}(A_i)$ */
2      Let $t = st_1 \cdots st_k \in \mathcal{L}(A_{\mathtt{f}}) \setminus \cup_{i=1}^{n} \mathcal{L}(A_i)$;
       /* Determine the status of $t$ (and update summaries $\mathcal{C}$) */
3      $R, \mathsf{path}(t) := Status(t, P, \neg Q)$;
       /* Status of $t$ is settled */
4      if $R$ then
           /* $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ is not valid and $\mathsf{path}(t)$ a counter-example */
5          return $\mathsf{path}(t)$;
6      else
           /* $t$ is $(P, \neg Q)$-infeasible, we refine and iterate */
7          Let $\mathcal{I} = I_0, \cdots, I_k \in \mathsf{itp}_{P,\neg Q}(t)$;
8          $n := n + 1$;
9          $A_n := A(P, \neg Q)^t_{\mathcal{I}}$;
10         $P_n := I_0$ and $Q_n := I_k$;
/* $\{P\}\ A_{\mathtt{f}}\ \{Q\}$ is valid. Add to $\mathcal{C}$ and returns a stronger summary */
11 $\mathcal{C}(\mathtt{f}) := \mathcal{C}(\mathtt{f}) \cup \{(\cap_{i=1}^{n} P_i, \cup_{i=1}^{n} Q_i))\}$;
12 return $\mathsf{itp}(\cap_{i=1}^{n} P_i, \cup_{i=1}^{n} Q_i)$;
```

$Status(t, P, \neg Q)$ is defined in Algorithm 3 and determines the $(P, \neg Q)$-feasibility status of a trace t, and in doing so may recursively call Algorithm 2 (line 8). Algorithm 3 determines the status of a trace $t = st_1 \cdot st_2 \ \cdots \ st_k$ as follows:

- function call statements are collected and stored into FCall (line 2). Then path is initialised with the default values for the statements that are not function calls (line 3).
- the $(P, \neg Q)$-feasibility status of t is determined in an iterative manner:
  - if t is $\mathcal{C}$-$(P, \neg Q)$-infeasible, the condition of line 5 is false and the else statement on line 16 is executed. This implies t is $(P, \neg Q)$-infeasible as $\mathcal{C}$ is an over-approximation, and we can return $(False, \bot)$.
  - if t is $\mathcal{C}$-$(P, \neg Q)$-feasible, we obtain some before/after witness values for the variables for each function call and store them in pairs $(\nu_i, \mu_i), i \in \mathsf{FCall}$. The for-loop at line 8 checks each function call w.r.t. to the feasibility of its before/after witness values. This is done by recursively calling $Hoare_2$ (Algorithm 2) on the callees by claiming that the witness assignment is not realisable by the function. The result of these recursive calls to $Hoare_2$ are either a witness trace $\mathsf{path}(u)$ or a pair of predicates $\mathsf{itp}(P', Q')$. If we get a witness trace we store it in $\mathsf{path}(st_i)$ (line 12), otherwise we do nothing (but the context $\mathcal{C}$ has been updated by the call to $Hoare_2$).

▶ Remark. One important feature of the algorithm to build the canonical interpolant automata [13, 15] is the ability to add back edges (thus defining loops) to the initial automaton that encodes the infeasible trace. An (back) edge labelled st can be added from

```
Algorithm 3: $Status(t, P, \neg Q)$
   Global : $\mathcal{C}$: context with summaries for each function
   Input  : A trace $t$, two predicates $P$ and $\neg Q$
   Result : $(False, \bot)$ if $t$ is $\mathcal{C}$-$(P, \neg Q)$-infeasible and thus $(P, \neg Q)$-infeasible
            $(True, \mathsf{path}(t))$ with $\mathsf{path}(t)$ a $(P, \neg Q)$-feasible full inter-procedural path.
1  Let $t = st_1 st_2 \cdots st_k$;
2  Let $\mathsf{FCall} = \{1 \leq i \leq k \mid st_i$ is a function call$\}$;
   /* Initialise path($t$) for regular statements */
3  foreach $i \in \{1, \cdots, k\} \setminus \mathsf{FCall}$ do $\mathsf{path}(st_i) := st_i$;
4  while $True$ do
5      if $\widehat{\mathsf{post}}(\mathcal{C}, t, P) \cap \neg Q \not\subseteq False$ then
           /* $t$ is $(P, \neg Q)$-feasible under $\mathcal{C}$ */
6          Let $\{(\nu_i, \mu_i), i \in \mathsf{FCall}\}$ be the set of witness before/after values;
           /* Check whether each function call step is feasible */
7          foreach $i \in \mathsf{FCall}$ do $\mathsf{path}(st_i) = \bot$;
8          foreach $i \in \mathsf{FCall}(t)$ do
9              Let $st_i$ be a call to f with f defined by f($\overline{x}$):$\overline{y}$ ;
10             switch $Hoare_2(\overline{x} = \nu_i, A_f, \neg(\overline{y} = \mu_i))$ do
11                 case $\mathsf{path}(u)$
                       /* $u$ s.t. $\mathsf{post}(u, \overline{x} = \nu_i) = \overline{y} = \mu_i$ */
12                     $\mathsf{path}(st_i) := \mathsf{path}(u)$;
13                 case $\mathsf{itp}(P', Q')$
                       /* f satisfies $\{\overline{x} = \nu_i\}$ $A_q$ $\{\neg(\overline{y} = \mu_i)\}$ */
                       /* $(P', Q')$ has been added to summary of f */
14         if $\bigwedge_{l \in \mathsf{FCall}} \mathsf{path}(st_l) \neq \bot$ then
15             return $(True, \mathsf{path}(t))$
16     else
           /* $t$ is $(P, \neg Q)$-infeasible under $\mathcal{C}$ and thus $(P, \neg Q)$-infeasible */
17         return $(False, \bot)$;
```

a location associated with an interpolant I to another associated with J if $\mathsf{post}(st, I) \subseteq J$. As the contexts contain only over-approximations for function calls we can safely check whether a back edge can be added or not. Checking whether $\mathsf{post}(st, I) \subseteq J$ still requires an SMT-solver even if we use inductive interpolants computed using weakest preconditions.

The following Theorem establishes the (partial) correctness of Algorithm 2:

▶ **Theorem 8.** *Let $\mathcal{C}$ be an initial over-approximation. If $Hoare_2(A_{\mathtt{f}}, P, Q)$ terminates and there are less than n calls to $Hoare_2$, then:*

a) *the result of $Hoare_2(A_{\mathtt{f}}, P, Q)$ is correct i. e.,*
   1. *if it returns $\mathsf{itp}(P', Q')$, $\{P'\}$ $A_{\mathtt{f}}$ $\{Q'\}$ holds, with $P \implies P'$, $Q' \implies Q$,*
   2. *if it returns $\mathsf{path}(t)$ then $\mathsf{path}(t)$ is a finite inter-procedural path and $\mathsf{post}(P, \mathsf{path}(t)) \cap \neg Q \not\subseteq False$,*

b) *during the computation $\mathcal{C}$ is always an over-approximation.*

Theorem 8 proves that $Hoare_2$ is sound by $a.1$). If the Hoare triple is not valid, and if the post operator is exact then the returned counter-example is also feasible by $a.2$). The algorithm is also trivially complete (as in [13]) relative to the existence of a modular Hoare proof for the non-recursive program: if a program is error-free, there exists a context $\mathcal{C}$ such that we can establish correctness.

The assumption that the post operator is exact for simple statements can be lifted while still preserving soundness. An over-approximation for post (e. g., for programs with non linear assignments, we can compute a linear over-approximation) ensures soundness. However, we may return a witness counter-example which is infeasible, and get some false positives.

Finally, as trace refinement is strictly more powerful [13] refinement-wise than predicate abstraction refinement, we obtain a modular inter-procedural analysis technique that is strictly more powerful than predicate abstraction refinement based modular inter-procedural analysis [2, 18].

6 Implementation and Experiments

We have implemented Algorithms 2 and 3 in a prototype iProc (written in Scala). The input language of iProc is a simple inter-procedural language and we use SMT-Interpol [9] as the back-end solver to check feasibility and generate inductive interpolants when needed. We have implemented our own algorithm to build interpolant automata. An initial Hoare triple $\{P\}$ `main` $\{Q\}$ is specified using `assume` and `assert` (e. g., Program P_1, Listing 1).

We have experimented with small test cases inspired from industrial case studies submitted by Goanna [8] customers and users. We focussed on array-out-of-bounds and NULL pointer dereferences detection as this can be encoded in our programming language with integers. We specifically analysed inter-procedural programs that were generating false positives with the tool Goanna. The results show that we correctly analyse all the programs removing all the false positives generated by the latest release of Goanna. This clearly demonstrates an improvement with regard to accuracy.

We are now building a more versatile version of our prototype to be able to parse and analyse C programs and properly demonstrate scalability compared to other tools. Extensions to support arrays and pointer aliasing [7] are currently being investigated.

7 Related Work

Algorithms for inter-procedural (data flow) analysis of imperative programs can be traced back to 1978 with the seminal work of Sharir and Pnueli [19], and later by Reps *et al.* [17]. However, practical techniques and tools have only been discovered in the last decade. Slam [4] is certainly the best known tool and has been successfully applied to large case studies (e. g., checking violations of API rules in device drivers.) It relies on powerful automated predicate abstraction refinement techniques. Blast [5] and CpaChecker [6] have been successfully applied to medium size projects. The previous tools perform predicate abstraction (and refinement) rather than trace abstraction (and refinement), and to the best of our knowledge they do not fully support modular inter-procedural computation of increasingly precise summaries, but rather perform (some form of) function call in-lining.

The use of interpolants to extract summaries has been the subject of some recent papers. Two approaches are close in spirit to our work: Whale [2] and FunFrog [18]. There are fundamental differences between the algorithm in Whale and FunFrog and ours. Whale builds *under-approximations* of functions which has a major drawback: an already computed summary is valid provided the summaries of other functions are valid; if it turns out that an existing summary is invalidated (which can happen as only an under-approximation of a function has been explored), all the *dependent* computed summaries are invalidated as well. FunFrog [18] is based on bounded model-checking and thus proves properties of functions *upto* a bounded unrolling for loops and recursive calls. The summaries computed by FunFrog are thus valid only for the bounded unrolling of the function (e. g., this might prevent this approach from discovering loop invariants). Moreover, the computation of the summaries themselves is not modular: if a trace is feasible using the currently available summaries, and must be further investigated, this is done by in-lining suspicious calls to check whether they are actually feasible. Smash [12] is another tool using function summaries. However, it relies on quantifier elimination to compute the summaries, which is expensive and performs parallel computation of *may* and *must* summaries which increases complexity. Saturn [1, 11] has been applied successfully to find bugs in the Linux kernel. It is summary-based but bottom-up, and the sizes of summaries are bounded in order to ensure termination. An extension of Saturn, Calysto [3] can extract counter-examples.

Finally, the intra-procedural trace refinement approach of [13] that we build on is implemented in Ultimate Automizer [20]. It has been extended to inter-procedural programs in [14]. The extension is very elegant and uses *nested words* to model inter-procedural traces and the corresponding notion of *nested interpolant automata* to prove partial correctness of recursive programs. However, it requires trace in-lining and thus is not modular. Designing a fully modular approach based on trace refinement is thus a challenge and the method we propose in this paper is a non obvious extension.

8 Conclusion

We have proposed a new algorithm which performs inter-procedural analysis in a fully modular way. Our algorithm extends the intra-procedural trace refinement algorithm of [13]. It analyses a function using available summaries for other functions and never performs any form of function or trace in-lining; it refines a function's summary each time the function is analysed; it is top-down, context-sensitive and provides a counter-example when a program is incorrect. We have implemented the algorithm in a prototype analysis tool. We have analysed small non-recursive programs inspired from industrial case studies that contain inter-procedural defects or are hard to prove correct. The results are promising, and the next step is to demonstrate that the approach is scalable, which we believe is the case. Indeed, we can easily obtain a parallel version of our algorithm, as checking whether each function call (line 8 in Algorithm 3) satisfies a pre/postcondition can be done concurrently. Our method is also robust and independent of the technique to establish the validity of Hoare triples. Any suitable analysis technique e. g., abstract interpretation, bounded model-checking, etc. can be used.

Finally, the method we presented also suggests we break existing functions into smaller parts (almost *out-lining*), e. g., sequences, while loops, if statements. This makes units to analyse smaller, increases independence and enables us to compute valid Hoare triples that are often re-usable like loop invariants.

Acknowledgments. The authors would like to thank the anonymous referees for their most valuable comments that helped in preparing the final version of this paper.

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1 Alex Aiken, Suhabe Bugrara, Isil Dillig, Thomas Dillig, Brian Hackett, and Peter Hawkins. An overview of the saturn project. In Manuvir Das and Dan Grossman, editors, *PASTE*, pages 43–48. ACM, 2007.

2 Aws Albarghouthi, Arie Gurfinkel, and Marsha Chechik. Whale: An interpolation-based algorithm for inter-procedural verification. In Viktor Kuncak and Andrey Rybalchenko, editors, *VMCAI*, volume 7148 of *LNCS*, pages 39–55. Springer, 2012.

3 Domagoj Babic and Alan J. Hu. Calysto: scalable and precise extended static checking. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *ICSE*, pages 211–220. ACM, 2008.

4 Thomas Ball, Vladimir Levin, and Sriram K. Rajamani. A decade of software model checking with SLAM. *Commun. ACM*, 54(7):68–76, 2011.

5 Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker Blast. *STTT*, 9(5-6):505–525, 2007.

6 Dirk Beyer and M. Erkan Keremoglu. CPAchecker: A Tool for Configurable Software Verification. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 184–190. Springer, 2011.

7 Sebastian Biallas, Mads Chr. Olesen, Franck Cassez, and Ralf Huuck. Ptrtracker: Pragmatic pointer analysis. In *SCAM*, pages 69–73. IEEE, 2013.

8 Mark Bradley, Franck Cassez, Ansgar Fehnker, Thomas Given-Wilson, and Ralf Huuck. High performance static analysis for industry. *ENTCS*, 289:3–14, 2012.

9 Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. SMTInterpol: An Interpolating SMT Solver. In Alastair F. Donaldson and David Parker, editors, *SPIN*, volume 7385 of *LNCS*, pages 248–254. Springer, 2012.

10 William Craig. Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. *J. Symb. Log.*, 22(3):269–285, 1957.

11 Isil Dillig, Thomas Dillig, and Alex Aiken. Sound, complete and scalable path-sensitive analysis. In Rajiv Gupta and Saman P. Amarasinghe, editors, *PLDI*, pages 270–280. ACM, 2008.

12 Patrice Godefroid, Aditya V. Nori, Sriram K. Rajamani, and SaiDeep Tetali. Compositional may-must program analysis: unleashing the power of alternation. In Hermenegildo and Palsberg [16], pages 43–56.

13 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Refinement of trace abstraction. In Jens Palsberg and Zhendong Su, editors, *SAS*, volume 5673 of *LNCS*, pages 69–85. Springer, 2009.

14 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Nested interpolants. In Hermenegildo and Palsberg [16], pages 471–482.

15 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 36–52. Springer, 2013.

16 Manuel V. Hermenegildo and Jens Palsberg, editors. *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*. ACM, 2010.

17 Thomas W. Reps, Susan Horwitz, and Shmuel Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 49–61, 1995.

18 Ondrej Sery, Grigory Fedyukovich, and Natasha Sharygina. Interpolation-based function summaries in bounded model checking. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *HVC*, volume 7261 of *LNCS*, pages 160–175. Springer, 2011.

19 Micha Sharir and Amir Pnueli. Two approaches to interprocedural data flow analysis. Technical report, 1978. Dpt. Of Computer Science, Courant Institute, NY, USA.

20 Ultimate Automizer. `http://ultimate.informatik.uni-freiburg.de/automizer/`.

A Two-Level Logic Approach to Reasoning about Typed Specification Languages

Mary Southern[1] and Kaustuv Chaudhuri[2]

1 University of Minnesota, USA
marys@cs.umn.edu

2 INRIA and LIX/École polytechnique, France
kaustuv.chaudhuri@inria.fr

Abstract

The *two-level logic approach* (*2LL*) to reasoning about computational specifications, as implemented by the *Abella* theorem prover, represents derivations of a *specification language* as an inductive definition in a *reasoning logic*. This approach has traditionally been formulated with the specification and reasoning logics having the *same* type system, and only the formulas being translated. However, requiring identical type systems limits the approach in two important ways: (1) every change in the specification language's type system requires a corresponding change in that of the reasoning logic, and (2) the same reasoning logic cannot be used with two specification languages at once if they have incompatible type systems. We propose a technique based on *adequate* encodings of the types and judgements of a typed specification language in terms of a simply typed higher-order logic program, which is then used for reasoning about the specification language in the usual *2LL*. Moreover, a single specification logic implementation can be used as a basis for a number of other specification languages just by varying the encoding. We illustrate our technique with an implementation of the *LF* dependent type theory as a new specification language for *Abella*, co-existing with its current simply typed higher-order hereditary Harrop specification logic, without modifying the type system of its reasoning logic.

1998 ACM Subject Classification F.4.1 Mathematical Logic: Proof theory

Keywords and phrases Abella theorem prover, two-level logic, typed specification languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.557

1 Introduction

Higher-order abstract syntax (*HOAS*) [13], also known as λ-*tree syntax* (λ*TS*) [8], has become a standard representational style for data structures with variable binding. Such data are pervasive in the syntax of programming languages, proof systems, process calculi, formalized mathematics, *etc.* Variable binding issues are a particularly tricky aspect of the meta-theory of computational systems given in the form of *structural operational semantics* (*SOS*). Such specifications are nearly always formulated as relations presented in the form of an inference system; for instance, the typing judgement for the simply typed λ-calculus is a relation between λ-terms and their types, usually defined in terms of a *natural deduction* proof system. Such relations on higher-order data can then be systematically formalized as higher-order logic programs in languages such as λ*Prolog* [9] or *Twelf* [14], which lets us directly animate the specifications by means of logic programming interpreters and compilers such as *Teyjus* [11].

In this paper, we are concerned with proving properties *about* such higher-order relational specifications. For example, if the specification is of the typing relation for simply typed λ-terms, then we may want to prove that a given λ-term has exactly one type (type-uniqueness), or that the type of a λ-term remains stable during evaluation (type-preservation). This kind of reasoning proceeds by induction on the derivations of the specified relations, so we need a formalism that supports both inductive definitions and reasoning by induction. The *two-level logic approach* (*2LL*) is a general scheme for such reasoning systems, where the *specification language* derivations are viewed as an inductively defined object in a *reasoning logic*. In this reasoning logic, the specification language derivations are given a *closed world reading*, which is to say that that derivability in the specification language is completely specified: it can not only establish that certain specification formulas or judgements are derivable, but also that others are *not* derivable, or that two specification derivations are (bi)similar. We focus on the *Abella* implementation of the *2LL*, which is an interactive tactics-based theorem prover designed to reason about a subset of higher-order *λProlog* programs seen as the logic of *higher-order hereditary Harrop* (*HOHH*) formulas [20, 19].

We consider an extension of the *2LL* that can use a single reasoning logic to reason about a number of different specification languages in the same development. The *HOHH* language has only simple types, which makes both the specifications and the reasoning somewhat verbose because structural invariants must be separately specified and explicitly invoked in theorems. Richer type systems can often encode such invariants intrinsically in the types; to illustrate, dependent types can be used to define a type of (representations of) well-typed λ-terms, which is not possible with just simple types. Moreover, with such richer type systems one can often use the inductive structure of the terms themselves to drive the inductive argument rather than using auxiliary relations.

Unfortunately, the *2LL* as currently defined [6] does not sufficiently address these desiderata. In particular, the specification and reasoning languages are required to have the same type system because the specification-level constants and their types are directly lifted to reasoning-level constants with the same type. Thus, if we required a version of *Abella* based on a dependent type theory as a specification language, we would need to also change its reasoning logic $\mathcal{G}$ to be dependently typed. This goes against the *2LL* philosophy where the reasoning logic is seen as common, static, and eternal. More importantly, it both breaks portability of developments and causes duplication of effort.

Our position is that we should extend the *2LL* in such a way that the reasoning-level and specification-level type systems are separated. Indeed, the specification types and judgements must themselves be encoded as terms and formulas of the reasoning logic. This encoding must be coherent with that of specification-level terms and formulas, both of which are encoded as reasoning-level atomic formulas. This is achieved by guaranteeing that our encoding of the type systems is *adequate*; that is, the encoding of the specification-level type system must be able to represent all specification-level typing derivations, *and* that reasoning about the specification-level type system should be reducible to reasoning, by induction, on the encoding. An essential ingredient of adequacy is a right-inverse of the encoding that extracts a specification-level typing judgement from a reasoning-level formula when the formula is in the image of the encoding.

To be concrete, we illustrate the extended *2LL* in this paper by giving an encoding of the *LF* dependent type theory, which is then implemented as a translation layer in *Abella*. The reasoning logic of *Abella* is left untouched, as is the existing *HOHH* specification language for reasoning about *λProlog* programs. Our encoding of *LF* is based on that of [17, 18], suitably modified to the context of interactive theorem proving rather than logic programming. Since both *LF* and *HOHH* are based on intuitionistic logic, our extension of *Abella* uses a core implementation of an intuitionistic specification language that is shared by both the *HOHH* and the *LF* languages. Interestingly, the details of the encoding into this core language can almost entirely be obscured for the user; in particular, to use the system the user does not need to know how the specification language is encoded, since the system uses the right inverse mentioned above to present the types, terms, and judgements of the specification language in their *native* forms.

The rest of the paper is organized as follows. Section 2 presents the two-level logic approach (*2LL*) and its implementation in the *Abella* theorem prover. Section 3 presents *LF* and its adequate translation into a simply typed higher-order logic programming language. This is then used in Section 4 to explain our extension of the *2LL* by means of adequate translations. Related work is surveyed in Section 5.

2 Background

This section sketches the two-level logic approach (*2LL*) as implemented in the *Abella* theorem prover [20]. More details, including the full proof systems and their meta-theory, can be found in the following sequence of papers: [19, 6].

2.1 The Reasoning Logic $\mathcal{G}$

The reasoning logic of *Abella*, $\mathcal{G}$, is a predicative and intuitionistic version of Church's Simple Theory of Types. Types are built freely from primitive types, which includes the type `prop` of formulas, using the function type constructor $\to$. Intuitionistic logic is introduced into this type system by means of the constants $\mathtt{true}, \mathtt{false} : \mathtt{prop}$, binary connectives $\wedge, \vee, \supset$: $\mathtt{prop} \to \mathtt{prop} \to \mathtt{prop}$, and an infinite family of quantifiers $\forall_\tau, \exists_\tau : (\tau \to \mathtt{prop}) \to \mathtt{prop}$ for types τ that do not contain `prop`. For every type τ not containing `prop`, we also add an atomic predicate symbol $=_\tau : \tau \to \tau \to \mathtt{prop}$ to reason about intensional (*i.e.*, up to $\alpha\beta\eta$-conversion) equality. Following usual conventions, we write $\wedge, \vee, \supset$, and $=_\tau$ infix, and write $\forall x : \tau.\, A$ for $\forall_\tau(\lambda x.\, A)$ (and similarly for $\exists$). We also omit the type subscripts and type-ascription on variables when unambiguous.

To provide the ability to reason on open λ-terms, which is necessary when reasoning about *HOAS* representations, $\mathcal{G}$ also supports *generic reasoning*. This is achieved by adding, for each type τ not containing `prop`, an infinite set of *nominal constants* and a generic quantifier $\nabla_\tau : (\tau \to \mathtt{prop}) \to \mathtt{prop}$. We also add a weaker form of intensional equality called *equivariance* that equates two terms whose free nominal constants may be systematically permuted to each other. Note that equivariance is only used to match conclusions to hypotheses in the $\mathcal{G}$ proof system; $=$ continues to have the standard λ-conversion semantics. The *support* of a term t, written $\mathrm{supp}(t)$, is the multiset of nominal constants that occur in it; whenever we introduce a new *eigenvariable*, such as using the $\forall$-right or $\exists$-left rules, we *raise* the eigenvariables over the support of the formula. This raising is needed to express permitted dependencies on these nominal constants.

To accommodate fixed-point definitions, $\mathcal{G}$ is parameterized by sets of *definitional clauses*. Each such clause has the form $\forall \vec{x}.\, (\nabla \vec{z}.\, A) \triangleq B$ where A (the *head*) is an atomic formula whose free variables belong to $\vec{x}$ or $\vec{z}$, while B (the *body*) is any arbitrary formula that can only mention the variables in $\vec{x}$, and can additionally have recursive occurrences of the predicate symbol in the head. Each clause partially defines a relation named by the predicate in the head. We additionally require that $\mathrm{supp}(\nabla \vec{z}.\, A)$ and $\mathrm{supp}(B)$ be both empty, and that recursive predicate occurrences satisfy a *stratification* condition [5].[1] Finally, some of

[1] Roughly, stratification prevents definitions such as $p \triangleq \neg p$, which would lead to inconsistency.

$$\begin{array}{rlrl}
\texttt{seq}\ L\ (G_1\ \&\ G_2) \triangleq \texttt{seq}\ L\ G_1 \wedge \texttt{seq}\ L\ G_2 & & \texttt{bch}\ L\ (F_1\ \&\ F_2)\ A \triangleq \texttt{bch}\ L\ F_1\ A \vee \texttt{bch}\ L\ F_2\ A \\
\texttt{seq}\ L\ (F \Rightarrow G) \triangleq \texttt{seq}\ (F :: L)\ G & & \texttt{bch}\ L\ (G \Rightarrow F)\ A \triangleq \texttt{seq}\ L\ G \wedge \texttt{bch}\ L\ F\ A \\
\texttt{seq}\ L\ (\texttt{pi}\ F) \triangleq \nabla x.\, \texttt{seq}\ L\ (F\ x) & & \texttt{bch}\ L\ (\texttt{pi}\ F)\ A \triangleq \exists t.\, \texttt{bch}\ L\ (F\ t)\ A \\
\texttt{seq}\ L\ A \triangleq \texttt{mem}\ F\ L \wedge \texttt{bch}\ L\ F\ A & & \texttt{bch}\ L\ A\ A \triangleq \texttt{true} \\
\texttt{seq}\ L\ A \triangleq \texttt{prog}\ F \wedge \texttt{bch}\ L\ F\ A & & &
\end{array}$$

Figure 1 Encoding *HOHH* using definitional clauses in $\mathcal{G}$. F and G range over arbitrary specification formulas, while A ranges over atomic specification formulas. All clauses are implicitly universally closed over their capitalized variables.

these definitions in $\mathcal{G}$ can be marked as *inductive* or *co-inductive*, in which case the set of definitional clauses for that relation are given least or greatest fixed-point semantics. This is approximated in *Abella* by means of *size annotations*, which are formally defined and proved correct in [4].

2.2 The Specification Language: *HOHH*

The essence of the *2LL* is to encode the deductive formalism of the specification language in terms of an inductive definition. However, before this can be done, the terms and formulas – and types! – of the specification language must be represented in the reasoning logic. This is trivial if the specification and reasoning logics have the same term and type language, which is the case for the *HOHH* language. To encode *HOHH* formulas, we use a new basic type `o`, and formula constructors $\Rightarrow, \& : \texttt{o} \to \texttt{o} \to \texttt{o}$ (written infix), and an infinite family of specification-level quantifiers $\texttt{pi}_\tau : (\tau \to \texttt{o}) \to \texttt{o}$ (standing for universal quantification, written prenex) for types τ that do not contain `o`. To prevent circularity, we disallow the type `prop` and the reasoning level formula constructors from occurring inside specification level types and terms.

The proof system for *HOHH* is a standard focused sequent calculus for this fragment of the logic, assuming that all atoms have negative polarity; this is equivalent to saying that the proof system implements *backchaining* [19]. This proof system is implemented in $\mathcal{G}$ using two predicates, $\texttt{seq} : \texttt{olist} \to \texttt{o} \to \texttt{prop}$ and $\texttt{bch} : \texttt{olist} \to \texttt{o} \to \texttt{o} \to \texttt{prop}$, standing for *goal reduction* and *backward chaining* respectively, with the definitional clauses shown in figure 1. Here, specification contexts have the type `olist`, the type of lists of `o`, with constructors $\texttt{nil} : \texttt{olist}$ and $(::) : \texttt{o} \to \texttt{olist} \to \texttt{olist}$ (written infix), and a membership relation $\texttt{mem} : \texttt{o} \to \texttt{olist} \to \texttt{prop}$ that has the obvious inductive definition. In *Abella*, these two relations are displayed using the more evocative notation $\{L \vdash G\}$ and $\{L, [F] \vdash G\}$ for $\texttt{seq}\ L\ G$ and $\texttt{bch}\ L\ F\ G$. The final clause for `seq` uses a separate predicate $\texttt{prog} : \texttt{o} \to \texttt{prop}$ that is true exactly for the clauses in the specification program. It is easy to see that with this syntax, the definitional clauses of figure 1 are precisely the inductive definition of a backchaining proof system.

2.3 Example: Type Uniqueness

The need for the two kinds of specification sequents and the mechanism for proving properties about the specification logic are best described with an example. Consider the simply typed λ-calculus, itself specified as an object logic in *HOHH*. The simple type system is represented using a new basic type `ty` with two constructors, $\texttt{i} : \texttt{ty}$ (a basic sort), and $\texttt{arr} : \texttt{ty} \to \texttt{ty} \to \texttt{ty}$ for constructing arrow types. The λ-terms are typed using a different basic type `tm` with two constructors: $\texttt{app} : \texttt{tm} \to \texttt{tm} \to \texttt{tm}$ and $\texttt{abs} : \texttt{ty} \to (\texttt{tm} \to \texttt{tm}) \to \texttt{tm}$. The λ-term $\lambda x{:}\texttt{i}.\, \lambda f{:}\texttt{i} \to \texttt{i}.\, f x$ would be represented as $\texttt{abs i}\ (\lambda x.\, \texttt{abs}\ (\texttt{arr i i})\ (\lambda f.\, \texttt{app}\ f\ x))$. The relation between terms (of type tm) and types (of type ty) is usually expressed in the form of an inference system such as:

$$\frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash (\lambda x{:}A.\, M) : A \to B}\to_I \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\to_E \qquad \frac{}{\Gamma, x{:}A \vdash x : A}\,\mathsf{var}$$

This relation is succinctly expressed as a pair of *HOHH* program clauses for the predicate $\texttt{of} : \texttt{tm} \to \texttt{ty} \to \texttt{o}$, which is used both for assumptions of the form $x{:}A$ and for conclusions of the form $M : A$ in the inference system above.

$$\texttt{pi}\ a{:}\texttt{ty}.\, \texttt{pi}\ b{:}\texttt{ty}.\, \texttt{pi}\ m{:}\texttt{tm}.\, \texttt{pi}\ n{:}\texttt{tm}.\, \texttt{of}\ m\ (\texttt{arr}\ a\ b) \Rightarrow \texttt{of}\ n\ a \Rightarrow \texttt{of}\ (\texttt{app}\ m\ n)\ b.$$

$$\texttt{pi}\ a{:}\texttt{ty}.\, \texttt{pi}\ b{:}\texttt{ty}.\, \texttt{pi}\ r{:}\texttt{tm} \to \texttt{tm}.\, (\texttt{pi}x{:}\texttt{tm}.\texttt{of}\ x\ \texttt{tm} \Rightarrow \texttt{of}\ (r\ x)\ \texttt{tm}) \Rightarrow \texttt{of}\ (\texttt{abs}\ a\ r)\ (\texttt{arr}\ a\ b).$$

Note that there is no clause for var; rather, it is folded into an assumption in the body of the abs case, which delimits its scope. It is generally easier to read such clauses when they are written using the standard *λProlog* syntactic convention of using capital letters for universally closed variables, writing implications in the reverse direction with the head first, and separating assumptions by commas rather than repeated implications. Thus, the above clauses correspond to:

```
of (app M N) B ⇐ of M (arr A B), of N A.
of (abs A R) (arr A B) ⇐ pi x\ of (R x) B ⇐ of x A.
```

The formula $\texttt{pi}\ (\lambda x.\, F)$ is rendered as `pi x\ F` in the concrete syntax, and the scope of `x` extends as far to the right as possible. Note that all the types are inferred.

In the reasoning mode of *Abella*, the above *λProlog* specification is *imported* by reflecting all specification constants and types in the reasoning signature, and by generating a definition for prog that is true only for the two clauses for of. The typing judgement $x{:}A, y{:}B \vdash M : C$ in the inference system (2.3), for instance, would be represented by $\texttt{seq}\ (\texttt{of}\ y\ B :: \texttt{of}\ x\ A :: \texttt{nil})\ (\texttt{of}\ M\ C)$. As an example of reasoning on this specification, we can prove that of is deterministic in its second argument:

```
forall M A B, {⊢ of M A} → {⊢ of M B} → A = B.
```

This theorem is proved by induction on the derivation of one of the seq assumptions, such as the first one. This induction would repeatedly *match* the form $\texttt{seq nil}\ M\ A$ against the left hand sides of the definitional clauses in figure 1; for every successful match, the corresponding right hand side of the clause would give us new assumptions, which may then be used in the inductive hypothesis.

Initially, the only clause that matches is the final one for seq corresponding to backchaining on a program clause. In the case where the clause for abs is selected, the corresponding bch clause for it would in turn call seq with a different list of assumptions. Thus, the inductive argument cannot proceed with empty dynamic specification contexts (the first argument to seq and bch) alone: we must also allow for reasoning under an abstraction. This is achieved in the reasoning logic by inductively characterizing all such dynamic context extensions with a new atom, say $\texttt{ctx} : \texttt{olist} \to \texttt{prop}$, with the following inductive definitional clauses:

$$\texttt{ctx nil} \triangleq \texttt{true}.$$

$$\forall A.\, \forall G.\, (\nabla x.\, \texttt{ctx}\ (\texttt{of}\ x\ A :: G)) \triangleq \texttt{ctx}\ G.$$

We can then prove a stronger lemma:

```
forall G M A B, ctx G → {G ⊢ of M A} → {G ⊢ of M B} → A = B.
```

$$\phi\,(\Pi x{:}A.\,P) := \phi\,(A) \to \phi\,(P) \qquad \langle c\rangle := c \qquad \langle \lambda x{:}A.\,M\rangle := \lambda x{:}\phi\,(A).\,\langle M\rangle$$
$$\phi\,(a\ M_1\ \cdots\ M_n) := \texttt{lfobj} \qquad \langle x\rangle := x$$
$$\phi\,(\texttt{type}) := \texttt{lftype} \qquad \langle M_1\ M_2\rangle := \langle M_1\rangle\ \langle M_2\rangle$$

Figure 2 Encoding of *LF* types and kinds as simple types and *LF* objects as simply typed λ-terms.

Now, when the dynamic context does grow when backchaining on the clause for `abs`, it will grow exactly by the form in the head of the second clause of `ctx`, *i.e.*, with a formula of the form `of` n A where n is a nominal constant that does not occur in A nor in the original context `G`. Thus, when we in turn backchain on the dynamic clauses (using the penultimate clause for `ctx` in figure 1), we will know the precise form of the selected clause.

3 An Adequate Translation of *LF* to *HOHH*

The Edinburgh Logical Framework (*LF*) is a dependently typed λ-calculus which is used for specifying formal systems. Terms of this language belong to one of the following three syntactic categories:

Kinds	$K ::= \texttt{type} \mid \Pi x{:}A.\,K$
Types	$A, B ::= a\ M_1 \ldots M_n \mid \Pi x{:}A.\,B$
Objects	$M, N ::= c \mid x \mid \lambda x{:}A.\,M \mid M\ N$

Types, sometimes called *families*, classify objects and kinds classify types. Here a represents a type-level constant, c an object level constant, and x an object level variable. Following standard convention, we will write $A \to B$ as a shorthand for $\Pi x{:}A.\,B$ when x does not appear free in B. We will use U to denote both types and objects and P for both kinds and types, so $U : P$ will stand either for a typing or a kinding judgement. We will write $U[M_1/x_1, \ldots, M_n/x_n]$ to denote the capture avoiding substitution of $M_1, \ldots, M_n$ for free occurrences of $x_1, \ldots, x_n$ respectively.

An *LF* specification is a list of object or type constants together with their types or kinds, called a *signature*. Let us revisit the example of the simply typed λ-calculus and its associated typing relation used in Section 2.3. The λ-terms are encoded using the following signature:

```
ty  : type.                   tm  : type.
i   : ty.                     app : tm → tm → tm.
arr : ty → ty → ty.           abs : ty → (tm → tm) → tm.
```

For the typing relation `of`, in *LF* we declare it as a dependent type rather than as a predicate as in *HOHH*. The clauses of the `of` type are then viewed as constructors for the dependent type, and are therefore also given names. Here, the concrete syntax `{x:A} B` denotes $\Pi x{:}A.\,B$.

```
of    : tm → ty → type.
ofApp : {A:ty} {B:ty} {M:tm} {N:tm}
          of M (arr A B) → of N A → of (app M N) B.
ofAbs : {A:ty} {B:ty} {R:tm → tm}
          ({x:tm} of x A → of (R x) B) → of (abs A R) (arr A B).
```

The *LF* type system is formally defined in [7] and will not be repeated here. Instead, we will directly give an adequate encoding of the *LF* type system in terms of *HOHH*, based on the variant of the encoding in [3] defined in [17], with the inverse mapping defined in [18].

The encoding proceeds in two steps. First, we transform our dependently typed terms into simply typed *HOHH* terms. The encoding of types and kinds is defined as a mapping, written $\phi\,(-)$. These types indicate that the term is an encoding of an *LF* type and an *LF* object, respectively. For each constant $c : P$ in the *LF* signature, we generate a simply typed

$$\begin{aligned}
\{\{\Pi x{:}A.\,P\}\} &:= \lambda m{:}\phi\,(\Pi x{:}A.\,P).\,\texttt{pi}\ x{:}\phi\,(A).\,\{\{A\}\}x \Rightarrow \{\{P\}\}(m\ x)\\
\{\{a\ M_1\ \cdots\ M_n\}\} &:= \lambda m{:}\texttt{lfobj}.\,\texttt{hastype}\ m\ (a\ \langle M_1\rangle\ \cdots\ \langle M_n\rangle)\\
\{\{\texttt{type}\}\} &:= \lambda m{:}\texttt{lftype}.\,\texttt{istype}\ m
\end{aligned}$$

Figure 3 Encoding of *LF* types and kinds using the `hastype` and `istype` predicates.

term c of type $\phi\,(P)$. Using this mapping, the dependently typed λ-terms are converted into simply typed λ-terms using the mapping $\langle - \rangle$. Figure 2 contains the rules for both $\langle - \rangle$ and $\phi\,(-)$. Note that $\phi\,(-)$ erases not just the type dependencies but also the identities of the types. For an atomic type $A = a\ M_1\ \cdots\ M_n$, we further write $\langle A\rangle$ to stand for $a\ \langle M_1\rangle\ \cdots\ \langle M_n\rangle$.

The second pass uses two new predicates, $\texttt{hastype} : \texttt{lfobj} \to \texttt{lftype} \to \texttt{o}$ and $\texttt{istype} : \texttt{lftype} \to \texttt{o}$, to encode the type and kind judgements of *LF*. Whenever $M : A$ is derivable in *LF* under a given signature, it must be the case that $\{\{A\}\}\langle M\rangle$ is derivable in *HOHH* from the clauses for `hastype` and `istype` produced from encoding the signature. Likewise, when $A : K$ is derivable, it should be the case that $\{\{K\}\}\langle A\rangle$ is derivable. The rules for this encoding are shown in figure 3.

▶ **Theorem 1** (Adequacy, [17]). *The LF hypothetical judgement $x_1 : P_1, \ldots, x_n : P_n \vdash M : A$ is derivable in the LF type theory [7] from an LF signature Σ if and only if the HOHH formula $\{\{P_1\}\}x_1 \Rightarrow \cdots \Rightarrow \{\{P_n\}\}x_n \Rightarrow \{\{A\}\}\langle M\rangle$ is derivable from the HOHH encoding of Σ according to the rules in figures 2 and 3.* ◀

Because this encoding is adequate, it is possible to define a right-inverse that maps a *HOHH* formula in the image of the translation in figure 3 back to an *LF* judgement. This inverse will be very useful in the next section where we will use the encoding of *LF* to extend the *2LL* via translations. The user of the system will not need to be aware of the details of the encoding as the *HOHH* formulas will be inverted into their corresponding *LF* judgements.

Defining such an inverse requires a small amount of care. We obviously cannot invert every *HOHH* formula, just those that correspond to a given signature. However, even for formulas constructed using the encodings of an *LF* signature, we may not necessarily be able to invert them; for instance, the formula may be the translation of a malformed or ill-typed *LF* judgement. This inverse will also not necessarily recover exactly the *LF* judgement used to construct the *HOHH* formula in the first place; rather, the inversion will only produce a unique inverse (if one exists) up to $\beta\eta$-conversion.

The inversion operation is defined in terms of the following four sequent forms:

$\Gamma \vdash \texttt{hastype}\ m\ a \longrightarrow M : A$	inverting typing; M, A output
$\Gamma \vdash \texttt{istype}\ a \longrightarrow A : \texttt{type}$	inverting kinding; A output
$\Gamma \vdash m : A \rightsquigarrow M$	inverting canonical terms; M output
$\Gamma \vdash m \rightsquigarrow M : A$	inverting atomic terms; M, A output

The rules are shown in figure 4. In each case, Γ contains the type and kind information for the signature constants and the typing assumptions for the bound variables in the input terms. A and B range over *LF* types, M and N over *LF* terms, F and G over *HOHH* formulas and a, m and n over simply typed λ-terms produced by $\langle - \rangle$. The rules for inverting typing and kinding are novel, but those for inverting terms are standard from bidirectional type-checking, and have already been developed in [18] (in a slightly more general form).

▶ **Theorem 2** (Right inverse). *If $\{\{P\}\}\langle U\rangle = F$ under the translation of Γ and $\Gamma \vdash F \longrightarrow U' : P$, then $\Gamma \vdash U =_{\beta\eta} U' : P$ in LF.*

$$\dfrac{(a{:}\Pi x_1{:}A_1.\cdots\Pi x_k{:}A_k.\,B)\in\Gamma \qquad \begin{array}{l}\Gamma\vdash m_1 : A_1 \rightsquigarrow M_1\\ \quad\cdots\\ \Gamma, x_1{:}A_1,\ldots,x_{k-1}{:}A_{k-1}\vdash m_k : A_k \rightsquigarrow M_k\\ \Gamma\vdash m : a\ M_1\ \cdots\ M_k \rightsquigarrow M\end{array}}{\Gamma\vdash \mathtt{hastype}\ m\ (a\ m_1\ \cdots\ m_k) \longrightarrow M : a\ M_1\ \cdots\ M_k}\ \textsf{inv-has}$$

$$\dfrac{(a{:}\Pi x_1{:}A_1.\cdots\Pi x_k{:}A_k.\,\mathtt{type})\in\Gamma \qquad \begin{array}{l}\Gamma\vdash m_1 : A_1 \rightsquigarrow M_1\\ \quad\cdots\\ \Gamma, x_1{:}A_1,\ldots,x_{k-1}{:}A_{k-1}\vdash m_k : A_k \rightsquigarrow M_k\end{array}}{\Gamma\vdash \mathtt{istype}\ (a\ m_1\ \cdots\ m_k) \longrightarrow a\ M_1\ \cdots\ M_k : \mathtt{type}}\ \textsf{inv-is}$$

$$\dfrac{\Gamma\vdash F\longrightarrow R : A \qquad R =_\eta x \qquad \Gamma, x{:}A\vdash G\longrightarrow M : B}{\Gamma\vdash \mathtt{pi}\ x{:}a.\,F \Rightarrow G\longrightarrow (\lambda x{:}A.\,M) : \Pi x{:}A.\,B}\ \textsf{inv-nest}$$

. . . .

$$\dfrac{\Gamma, x{:}A\vdash m : B\rightsquigarrow M}{\Gamma\vdash(\lambda x{:}T.\,m) : (\Pi x{:}A.\,B)\rightsquigarrow \lambda x{:}A.\,M}\ \textsf{inv-lam} \qquad \dfrac{\Gamma\vdash m\rightsquigarrow M : \Pi x{:}A.\,B \qquad \Gamma\vdash n : A\rightsquigarrow N}{\Gamma\vdash m\ n\rightsquigarrow M\ N : B[N/x]}\ \textsf{inv-app}$$

$$\dfrac{\Gamma\vdash m\rightsquigarrow M : A}{\Gamma\vdash m : A\rightsquigarrow M}\ \textsf{inv-switch} \qquad \dfrac{(x{:}A)\in G}{\Gamma\vdash x\rightsquigarrow x : A}\ \textsf{inv-hyp} \qquad \dfrac{(c{:}A)\in G}{\Gamma\vdash c\rightsquigarrow c : A}\ \textsf{inv-const}$$

Figure 4 Inverting the *LF* encoding of judgements ($\longrightarrow$) and terms ($\rightsquigarrow$).

Proof. By structural induction on the inversion derivation. Note the requirement for η-contraction of the term to a variable in the second premise of inv-nest is necessary, for otherwise the rule would produce an unsound abstraction. If the formula F was generated from the translation of figures 2 and 3, then this η-contraction check will always succeed. ◀

4 Translational Two-level Logic Approach

We will now use both the translation of *LF* signatures to *HOHH* formulas and its inverse to extend the *2LL* in such a way that we can reason about *LF* signatures just as we were able to reason on *λProlog* specifications as shown in the example of Section 2.3.

4.1 Importing the *LF* Specification

As the type system of *LF* and $\mathcal{G}$ are different, we cannot directly reflect the constants and types of the *LF* specification logic like we did with *HOHH* in Section 2. Instead, for every *LF* constant c of *LF* type or kind P in the *LF* signature, we do the following: (1) add a constant $c : \phi\,(P)$ to the $\mathcal{G}$ signature; and (2) add the clause $\{\{P\}\}c$ to the `prog` definition. Note that because the clauses are always of the form $\{\{P\}\}c$ for a constant c, there will never be any redexes in the clauses, *i.e.*, the generated clauses are β-normal. They are also η-long, because the definition of $\{\{-\}\}$ traverses the type or kind until it is atomic.

▶ **Example 3.** Consider again the *LF* signature in Section 3. When it is imported into $\mathcal{G}$, the following constants are added to the $\mathcal{G}$ signature by step (1):

```
ty   : lftype.                     tm  : lftype.
i    : lfobj.                      app : lfobj → lfobj → lfobj.
arr  : lfobj → lfobj.              abs : lfobj → (lfobj → lfobj) →
    lfobj.

of     : lfobj → lfobj → lftype.
ofApp  : lfobj → lfobj → lfobj → lfobj → lfobj → lfobj → lfobj.
ofAbs  : lfobj → lfobj → (lfobj → lfobj) →
             (lfobj → lfobj → lfobj) → lfobj.
```

The following clauses are then added to `prog` by step (2) described above:

```
lfisty ty.
lfhas i ty.
lfhas (arr Z1 Z2) ty ⇐ lfhas Z1 ty, lfhas Z2 ty.

lfisty tm.
lfhas (app Z1 Z2) tm ⇐ lfhas Z1 tm, lfhas Z2 tm.
lfhas (abs Z1 Z2) tm ⇐
  lfhas Z1 ty, (pi x\ lfhas x tm ⇒ lfhas (Z2 x) tm).

lfisty (of Z1 Z2) ⇐ lfhas Z1 tm, lfhas Z2 ty.
lfhas (ofApp A B M N Z1 Z2) (of (app M N) B) ⇐
  lfhas A ty, lfhas B ty, lfhas M tm, lfhas N tm,
  lfhas Z1 (of M (arr A B)), lfhas Z2 (of N A).
lfhas (ofAbs A B R Z1) (of (abs A (x\ R x)) (arr A B)) ⇐
  lfhas A ty, lfhas B ty,
  (pi x\ lfhas x tm ⇒ lfhas (R x) tm),
  (pi x\ lfhas x tm ⇒ pi z\ lfhas z (of x A) ⇒
            lfhas (Z1 x z) (of (R x) B)).
```

The variables named $\mathtt{Z}_i$ are generated by the translator for those variables that are omitted from the input signature by the use of $\rightarrow$ instead of Π. We write clauses using standard λ*Prolog* syntax for clarity; it is simple to take the output of translation into this form.

It is instructive to compare these clauses to those for the pure *HOHH* version in Section 2.3. Although, on the surface, these two look quite different, there are similarities in the kinds of subgoals that are produced for the three constructors of `of`. For example, consider the case of `ofApp`. Two of the formulas, $\mathtt{hastype}\ Z_1\ (\mathtt{of}\ M\ (\mathtt{arr}\ A\ B))$ and $\mathtt{hastype}\ Z_2\ (\mathtt{of}\ N\ A)$ are already present in nearly this form in the *HOHH* specification. The additional assumptions are just repetitions of the typing assumptions for the arguments to `ofApp`; indeed, many of them are redundant since the `ofApp` term is already assumed to be type-correct. This kind of redundancy analysis can be used to further improve the translation, making it nearly identical to the simply typed specification [17, 18].

4.2 Representing *LF* Hypothetical Judgements

We use the concrete syntax $\langle M : A\rangle$ or $\langle A : K\rangle$ to depict $\{\{A\}\}M$ or $\{\{K\}\}A$, respectively. In fact, since the *LF* type system is given in terms of hypothetical derivations, we generalize this syntax to the form: $\langle x_1 : P_1, ..., x_n : P_n \vdash U : P\rangle$ as an abbreviation for: $\mathtt{seq}\ (\langle x_1 : P_1\rangle :: \cdots :: \langle x_n : P_n\rangle)\ (\langle U : P\rangle)$. As an example,[2] the uniqueness theorem for the `of` relation is (eliding types):

$$\forall G, M, A, B, P_1, P_2, \mathtt{ctx}\ G \supset \langle G \vdash P_1 : \mathtt{of}\ M\ A\rangle \supset \langle G \vdash P_2 : \mathtt{of}\ M\ B\rangle \supset A = B. \tag{1}$$

Here, P_1 and P_2 are (encodings of) the *LF* proof-terms for the judgements $\mathtt{of}\ M\ A$ and $\mathtt{of}\ M\ B$ respectively; these proof terms are built out of the constructors for the `of` relation, *viz.* `ofApp` and `ofAbs`.

Of course, in order to prove this theorem we would require a suitable `ctx` definition. Unlike in the simply typed case, the recursive case for λ-abstractions not only introduces a new variable but also a proof that it has a given *LF* type at the same time. This gives us the following definitional clauses.

$$\mathtt{ctx}\ \mathtt{nil} \triangleq \mathtt{true}.$$

$$\nabla x{:}\mathtt{lfobj}.\ \nabla p{:}\mathtt{lfobj}.\ \mathtt{ctx}\ (\langle x : \mathtt{tm}\rangle :: \langle p : \mathtt{of}\ x\ A\rangle :: G) \triangleq \mathtt{ctx}\ G.$$

[2] The full *Abella*/*LF* development may be interactively browsed online at `http://abella-prover.org/lf`.

It is interesting to note that, because variables are introduced (bound) in a different place than their typing assumptions, it would be just as valid to use the following clause instead for the second clause above:

$$\nabla x{:}\mathtt{lfobj}.\,\nabla p{:}\mathtt{lfobj}.\,\mathtt{ctx}\ (\langle p : \mathtt{of}\ x\ A\rangle :: \langle x : \mathtt{tm}\rangle :: G) \triangleq \mathtt{ctx}\ G.$$

This reordering of the context that is not strictly allowed in the *LF* type system poses no problems for us. Indeed, when we reason about the elements of the context, we can always recover these two assumptions that are always simultaneously added to the context.

```
    forall G, nabla p x,
      ctx (G x p) → mem ⟨x : tm⟩ (G x p) →
        exists A, mem ⟨p : of x A⟩ (G x p) ∧ fresh p A ∧ fresh x A.
```

The dependency of `G` on `x` and `p` is indicated explicitly using application. For `A`, this dependency is implicit, because the `exists` occurs in the scope of the corresponding `nabla`s, so we use the predicate `fresh` : `lfobj` → `lfobj` → `prop` to further assert that its first arguments are nominal constants that do not occur in its second arguments. This is definable with the single clause: $\forall A.\,(\nabla x.\,\mathtt{fresh}\ x\ A) \triangleq \mathtt{true}$.

The proof of (1) proceeds by induction on the second assumption, $\langle G \vdash P_1 : \mathtt{of}\ M\ A\rangle$, using the clauses added to `prog` when importing the specification. There are exactly three backchaining possibilities for `prog` clauses, corresponding to the `ofApp` and `ofAbs` cases, respectively. Finally, when backchaining on the dynamic clauses in `G`, we use the `ctx` definition to characterize the shape of the selected clause: if the selected clause is $\langle x : \mathtt{tm}\rangle$, then the branch immediately succeeds since $\langle x : \mathtt{tm}\rangle$ will not unify with $\langle P_1 : \mathtt{of}\ M\ A\rangle$. Thus, the only backchaining case worth considering is when the selected dynamic clause is of the form $\langle p : \mathtt{of}\ x\ A'\rangle$. In this case, we continue by case-analysis of the second derivation, $\langle P_2 : \mathtt{of}\ M\ B\rangle$, in which case again the only possibility that is not immediately ruled out by unification is the case of $\langle p' : \mathtt{of}\ x\ B'\rangle$ being selected from G. In this case, we appeal to a *uniqueness lemma* [1] of the following form:

$$\forall G, X, A, B, P_1, P_2, \mathtt{ctx}\ G \supset \mathtt{mem}\ \langle P : \mathtt{of}\ X\ A\rangle\ G \supset \mathtt{mem}\ \langle P_2 : \mathtt{of}\ X\ B\rangle\ G \supset A = B.$$

The rest of the proof is fairly systematic, and largely identical in structure to that of the *HOHH* case. It is also worth remarking that once we have shown that the types A and B are identical in (1), we can then also show that the proof terms P_1 and P_2 must also be equal (up to $\alpha\beta\eta$, of course).

```
  forall G M A B P1 P2, ctx G →
    ⟨G ⊢ P1 : of M A⟩ → ⟨G ⊢ P2 : of M B⟩ → P1 = P2.
```

This is expected from the *LF* type theory, but would be difficult to state in *LF* itself because of the lack of equality as a built-in relation.

4.3 The Implementation

The implementation of the translational *2LL* can be found in the `lf` branch of the *Abella* repository.[3] This implementation also comes with a few examples of reasoning on *LF* specifications that can be browsed online without needing to run *Abella*. We have made the following observations about these developments:

[3] Details for downloading and building this branch can be found in `http://abella-prover.org/lf`.

- The user of the system never needs to look at the encoding of *LF* in *HOHH* directly. The system always translates *LF* judgements, written using $\langle - \rangle$, transparently to *HOHH*, and also inverts any *HOHH* formulas in the image of the translation back into an *LF* (hypothetical) judgement. Hence, the only domain knowledge the user needs to use the system is the tactics-based proof language of *Abella* itself.
- Our implementation currently does not perform type-checking on the *LF* judgements written by the user, either in the specification itself or as part of reasoning. This is not as such a problem, since we can never prove anything false about well-typed judgements. However, without type checking we have no way to verify that the theorem which has been proved is really meaningful since we are allowed to reason about ill-formed *LF* judgements. It would also be useful for users to have a type-checker as a sanity check. For the time being, we run the input specification through the *Twelf* system [14], both to type-check it and to get an explicit form of the specification.

5 Related Work and Conclusion

We have proposed here a translational extension to the two-level logic approach for reasoning about specifications. By adding a translation layer to the *Abella* theorem prover we have been able to reason over dependently typed *LF* specifications without needing to change the reasoning logic, and allowing *LF* to co-exist with the *HOHH* specification logic. We are already in the process of extending this implementation to arbitrary pure type systems instead of just *LF*; in particular, extending the type system with polymorphism, which is the most common feature request for *Abella*, should be encodable via our translation that realizes specification types as reasoning terms.

The translation of *LF* to *HOHH* used in this work is a minor variant of the *simple* translation from [17], which is itself based on earlier work [3], while the inversion on terms is similar to the definition in [18], omitting meta-variables. Various optimized versions of this translation have been used to use λ*Prolog* as an engine for logic programming with *LF* specifications; in particular, the *Parinati* system [17] and its extension to meta-variables in [18]. The meta-theory of the optimized translation is not as immediate as for the simple translation, but it would be interesting to investigate its use for the *Abella*/*LF* variant in the future. The combination of *Parinati* and *Abella*/*LF* gives us both an efficient execution model for dependently typed logic programs and a mechanism to reason about the meta-theory of such specifications in the extended *2LL*. In effect, *LF* becomes as much a first class citizen of the *Abella* ecosystem as *HOHH* and λ*Prolog* have traditionally been.

There are many other systems designed to reason with and about *LF* specifications. The most mature implementation is *Twelf* [14], which has a very efficient type-checker incorporating sophisticated term and type reconstruction. As mentioned in Section 4.3, we use *Twelf* to type-check and elaborate the *LF* specifications we import in *Abella*. In addition to the type-checker, *Twelf* has a suite of meta-theoretic tools that can verify certain properties of *LF* specifications, such as that a declared relation determines a total function. *Twelf* is, however, not powerful enough to reason inductively on arbitrary *LF* derivations. For example, although *Twelf* can check coverage, it cannot express the logical formula that corresponds to the coverage property.

Some of these expressive deficiencies of *Twelf* have been addressed in the *Delphin* [16] and *Beluga* [15] systems that add a functional programming language that can manipulate and reason inductively on *LF* syntax. The *Beluga* system, in particular, extends the *LF* type theory with *contextual modal types* [12] that give a type-theoretic treatment for meta-variables and explicit substitutions; in addition, *Beluga* also allows abstraction over contexts and substitutions [2]. The type-checker of *Beluga* is therefore very sophisticated and performs many kinds of reasoning on contexts automatically that must be done manually in *Abella*. On the flip-side, *Abella* has a small trusted core based on the logic $\mathcal{G}$ with a well-understood and – importantly! – stable proof system [10, 5]. It would be interesting to formally compare the representational abilities of *Abella*/LF and *Beluga*. Moreover, *Abella* has recently acquired a Plugin architecture that allows arbitrary (but soundness-preserving) user-written extensions to its automation capabilities [1], which might help us add more automation in the future.

References

1 Olivier Savary Bélanger and Kaustuv Chaudhuri. Automatically deriving schematic theorems for dynamic contexts. In *LFMTP'14*, pages 9:1–9:8. ACM, 2014.

2 Andrew Cave and Brigitte Pientka. Programming with binders and indexed data-types. In *POPL*, pages 413–424. ACM, 2012.

3 Amy Felty and Dale Miller. Encoding a dependent-type λ-calculus in a logic programming language. In *CADE*, volume 449 of *LNAI*, pages 221–235. Springer, 1990.

4 Andrew Gacek. *A Framework for Specifying, Prototyping, and Reasoning about Computational Systems*. PhD thesis, University of Minnesota, 2009.

5 Andrew Gacek, Dale Miller, and Gopalan Nadathur. Nominal abstraction. *Information and Computation*, 209(1):48–73, 2011.

6 Andrew Gacek, Dale Miller, and Gopalan Nadathur. A two-level logic approach to reasoning about computations. *J. of Automated Reasoning*, 49(2):241–273, 2012.

7 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

8 Dale Miller and Gopalan Nadathur. A computational logic approach to syntax and semantics. 10th Symp. of the Mathematical Foundations of Computer Science, May 1985.

9 Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.

10 Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, October 2005.

11 Gopalan Nadathur and Dustin J. Mitchell. System description: Teyjus – A compiler and abstract machine based implementation of λProlog. In *CADE*, number 1632 in Lecture Notes in Artificial Intelligence, pages 287–291. Springer, 1999.

12 Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual model type theory. *ACM Trans. on Computational Logic*, 9(3):1–49, 2008.

13 Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *PLDI*, pages 199–208. ACM Press, June 1988.

14 Frank Pfenning and Carsten Schürmann. System description: Twelf – A meta-logical framework for deductive systems. In *16th Conf. on Automated Deduction (CADE)*, number 1632 in LNAI, pages 202–206, Trento, 1999. Springer.

15 Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *IJCAR*, pages 15–21. Springer LNCS 6173, 2010.

16 Adam Poswolsky and Carsten Schürmann. System description: Delphin – A functional programming language for deductive systems. In *LFMTP*, volume 228, pages 113–120, 2008.

17 Zachary Snow, David Baelde, and Gopalan Nadathur. A meta-programming approach to realizing dependently typed logic programming. In *PPDP*, pages 187–198, 2010.

18 Mary Southern and Gopalan Nadathur. A λProlog based animation of Twelf specifications, July 2014. Available at `http://arxiv.org/abs/1407.1545`.

19 Yuting Wang, Kaustuv Chaudhuri, Andrew Gacek, and Gopalan Nadathur. Reasoning about higher-order relational specifications. In *PPDP*, pages 157–168, Madrid, Spain, September 2013.

20 The Abella web-site. `http://abella-prover.org/`, 2013.

On the Complexity of Computing Maximum Entropy for Markovian Models

Taolue Chen[1] **and Tingting Han**[2]

1 **Department of Computer Science, Middlesex University London, UK**
2 **Department of Computer Science and Information Systems, Birkbeck, University of London, UK**

Abstract

We investigate the complexity of computing entropy of various Markovian models including Markov Chains (MCs), Interval Markov Chains (IMCs) and Markov Decision Processes (MDPs). We consider both entropy and entropy rate for general MCs, and study two algorithmic questions, i.e., entropy *approximation* problem and entropy *threshold* problem. The former asks for an approximation of the entropy/entropy rate within a given precision, whereas the latter aims to decide whether they exceed a given threshold. We give polynomial-time algorithms for the approximation problem, and show the threshold problem is in $\mathsf{P}^{\mathsf{CH}_3}$ (hence in PSPACE) and in P assuming some number-theoretic conjectures. Furthermore, we study both questions for IMCs and MDPs where we aim to *maximise* the entropy/entropy rate among an infinite family of MCs associated with the given model. We give various conditional decidability results for the threshold problem, and show the approximation problem is solvable in polynomial-time via convex programming.

1998 ACM Subject Classification G.3 Probability and Statistics, D.2.4 Software/Program Verification

Keywords and phrases Markovian Models, Entropy, Complexity, Probabilistic Verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.571

1 Introduction

Entropy is one of the most fundamental notions in information theory which usually refers to the *Shannon entropy* in this context [16]. In a nutshell, it is the expected value of the information contained in a message. Markovian processes and entropy are related since the introduction of entropy by Shannon. In particular, Shannon defined and studied technically the *entropy rate* of a *discrete-time Markov chain* (henceforth MC in short) with a finite state space, which is one of the main topics of the current paper.

We identify two types of "entropy" defined in literature for MCs. Essentially entropy is a measure of uncertainty in random variables, and MCs, as a stochastic process, are a sequence of random variables. Naturally this view yields two possible definitions, intuitively the "average" and the "sum" of the entropy of the random variables associated with the MC, respectively:

- the classical definition of entropy, dating back to Shannon, typically known as the *entropy rate*. Informally, this is the time density of the *average* information in a stochastic process. *Henceforth, we refer to this definition as entropy rate.*
- the definition given by Biondi *et al* [7], which is the joint entropy of the (infinite) sequence of random variables in a stochastic process. Although being infinite in general, the authors argue that this represents, for instance, the information leakage where the states

of the MC are the observables of a deterministic program [7]. *Henceforth, we refer to this definition as entropy.*

Formal accounts are given in Section 3. Definitions of entropy of MCs raise algorithmic challenges. One natural question is, given an MC, how to "compute" its entropy? Note that in general, it is *not* a rational (even not an algebraic) number, which prompts the question what computing means exactly. Technically there are (at least) two possible interpretations which we formulate as the *entropy approximation problem* and the *entropy threshold problem*, respectively. Let $\mathcal{D}$ be an MC and $\hbar$ denote the entropy/entropy rate of $\mathcal{D}$.

- The entropy approximation problem aims to compute, given the error bound $\epsilon > 0$, a rational number θ such that $|\hbar - \theta| \leq \epsilon$;
- The entropy threshold problem aims to decide, given the rational number θ, whether $\hbar \bowtie \theta$, where $\bowtie \in \{<, \leq, =, \geq, >\}$.

Observe that general speaking the approximation problem is no harder than the threshold problem, since it can be solved by a simple binary search with the threshold problem as the oracle. However, the converse does *not* hold in general.

On top of a purely probabilistic model like MCs, it is probably more interesting to consider probabilistic models with *nondeterminism*, typically Interval Markov chains (IMCs) and Markov Decision Processes (MDPs). MDPs [26] are a well-established model which is widely used in, for instance, robotics, automated control, economics, and manufacturing. IMCs [22] are MCs where each transition probability is assumed to be within a range (interval). They are introduced to faithfully capture the scenario where transition probabilities are usually estimated by statistical experiments and thus it is not realistic to assume they are exact.

By and large, a probabilistic model with nondeterminism usually denotes an (infinite) family of pure probabilistic models. Among these models, selecting the one with the *maximum* entropy is one of the central questions in information theory [16]. As before, it raises algorithmic challenges as well, i.e., given an IMC or MDP which denotes an infinite family of MCs, how to "compute" the *maximum entropy*? Note the dichotomy of the approximation and the threshold problem exists here as well, which we shall refer to the *maximum entropy approximation problem* and the *maximum entropy threshold problem*, respectively.

Entropy of probabilistic models has a wide range of applications, in particular in security [12, 6, 30]. As a concrete example which is one of the motivations of the current paper, in a recent paper [7], all possible attacks to a system are encoded as an IMC, and the channel capacity computation reduces to finding an MC with highest entropy. Note that tool support has been already available [8].

Contributions. In this paper we are mainly interested in the algorithmic aspects of entropy for Markovian models. In particular, we carry out a theoretical study on the complexity of computing (maximum) entropy for MCs, IMCs, and MDPs. The main contributions are summarised as follows:

1. We consider the definition of entropy rate for general (not ergodic) MCs, and give a characterisation in terms of local entropy;
2. We identify the complexity of the entropy approximation problem and the entropy threshold problem for MCs;
3. We identify the complexity of the approximation problem for maximum entropy/entropy rate for IMCs, and we obtain *conditional* decidability for the threshold problem. These results can be adapted to the MDP model as well.

The main results of the paper are summarised in Table 1.

Table 1 Complexity of computing entropy/entropy rate

	approximation	threshold
MC	P	$\mathsf{P}^{\mathsf{CH}_3}$ (conditional in P)
IMC/MDP	P	conditional decidable

Some remarks are in order:

- Regarding **1**, in literature entropy rate is defined exclusively over *irreducible* (sometimes called ergodic) MCs where the celebrated Shannon-McMillan-Breiman theorem [16] actually gives a characterisation in terms of stationary distribution and local entropy. However, for computer science applications, MC models are seldom irreducible. Hence we provide a characterisation for general (finite-state) MCs, inspired by the one in [7].
- For the "computation" of entropy of MCs, [7] states that it can be done in polynomial time. Although not stated explicitly, this actually refers to the approximation problem. The threshold problem is not addressed in [7], nor the corresponding problems wrt. the entropy rate.
- For the "computation" of maximum entropy of IMCs, [7] considers the approximation problem. The authors reduce the problem to non-linear programming (over a convex polytope though) to which no complexity result is given. Here, instead, we show, by reducing to *convex programming*, the approximation problem can be solved in polynomial time. Note that the formulation in [7] is not convex in general, so we cannot start from there straightforwardly.
- For maximisation of entropy rate, it is actually a classical topic for MCs and semi-MCs. A classical result, due to Parry [24], shows how to define a (stationary) MC (called Shannon-Parry MC) over a given strongly connected graph to achieve the maximum entropy rate. More recent results focus on finding a (semi-)MC with the maximum entropy rate when its stationary distribution is constrained in certain ways, see, e.g., [19]. In contrast, here we work on the entropy rate for general IMCs and MDPs. To the best of our knowledge this is the first work of this type.

Related work. Apart from the work we have discussed before, [30, 12] studied the complexity of quantitative information flow for boolean and recursive programs, whereas [11] studied the information-leakage bounding problem (wrt. Shannon entropy) for deterministic transition systems. [9] studied entropy in process algebra. These models and questions are considerably different from ours. [13, 27, 15, 25, 4] studied IMCs and their model checking problems. The technique to solve convex programming is inspired by [25]. We also mention that [2] generalised Parry's result to the graph generated by timed automata.

An extended version of the paper [14] contains proofs, detailed expositions, and in particular, all results for MDPs.

2 Preliminaries

Let $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ denote the set of natural, rational, real numbers, respectively. Given any finite set S, we write $\Delta(S)$ for the set of *probabilistic distributions* over S, i.e., functions $\mu : S \to [0,1]$ with $\sum_{s\in S} \mu(s) = 1$. For any vector $\vec{x}$, we write $\vec{x}_i$ for the entry of $\vec{x}$ corresponding to the index i, and $\vec{x} \geq 0$ if $\vec{x}_i \geq 0$ for each i. Throughout this paper, $X, Y, \cdots$ denote discrete *random variables* (RVs), usually over a finite set of outcomes. For the RV X, we often denote the set of outcomes as $\mathcal{X} = \{x_1, \cdots, x_n\}$ which is ranged over by x. In this context, we also write $\Pr(X = x)$ or simply $p(x)$ for the *probability mass function.*

2.1 (Interval) DTMCs

▶ **Definition 1** (MC). A *(discrete-time) Markov chain* (MC) is a tuple $\mathcal{D} = (S, \alpha, \mathbf{P})$, where S is a finite set of *states*; $\alpha \in \Delta(S)$ is the *initial distribution*; and $\mathbf{P} : S \times S \to [0, 1]$ is the *transition probability matrix*, satisfying $\forall s \in S$, $\sum_{s' \in S} \mathbf{P}(s, s') = 1$.

Alternatively, an MC can be defined as a stochastic process $\{X_n\}_{n \geq 0}$, where each X_n is a discrete RV over S. The process respects the Markov property, i.e., $\Pr(X_n = s_n | X_{n-1} = s_{n-1}, \cdots, X_0 = s_0) = \Pr(X_n = s_n | X_{n-1} = s_{n-1}) = \mathbf{P}(s_{n-1}, s_n)$ for any $s_0, s_1, \cdots, s_n \in S$ and $n \in \mathbb{N}$. Note that $\Pr(X_n = s)$ denotes the probability of being in state s at time n. The *transient distribution* of $\mathcal{D}$ is denoted by $\pi^{(n)} \in \Delta(S)$, which can be computed by $\pi^{(n)} = \alpha \mathbf{P}^n$. It is known that $\Pr(X_n = s) = \pi_s^{(n)}$.

For a finite MC, we often use graph-theoretical notations which refer to the underlying digraph of $\mathcal{D}$. Essentially the vertices of the digraph are states of $\mathcal{D}$, and there is an edge from s to t iff $\mathbf{P}(s, t) > 0$. The following notions are standard.

▶ **Definition 2.**
- A subset $T \subseteq S$ is *strongly connected* if for each pair of states $s, t \in T$, t is reachable from s. A *strongly connected component* (SCC) T of an MC $\mathcal{D}$ denotes a strongly connected set of states such that no proper superset of T is strongly connected.
- A *bottom strongly connected component* (BSCC) T is an SCC from which no state outside T is reachable.

We write $\mathcal{E}(\mathcal{D})$ for the set of all SCCs of $\mathcal{D}$ and $\mathcal{B}(\mathcal{D}) \subseteq \mathcal{E}(\mathcal{D})$ for the set of all BSCCs of $\mathcal{D}$.

▶ **Definition 3.**
- A state s is *absorbing* if $\mathbf{P}(s, s) = 1$, i.e. s contains only a self-loop. An MC is *absorbing* if every state can reach an absorbing state.
- A state s is *transient* if, starting in state s, there is a non-zero probability that it will never return to s; otherwise s is *recurrent.*
- A state s is *deterministic* if the distribution $\mathbf{P}(s, \cdot)$ is Dirac, i.e. there is a unique t such that $\mathbf{P}(s, t) = 1$; otherwise s is *stochastic.*
- An MC is *irreducible* if its underlying digraph is strongly connected.

▶ **Definition 4** (IMC). An *interval-valued (discrete-time) Markov chain* (IMC) is a tuple $\mathcal{I} = (S, \alpha, \mathbf{P}^l, \mathbf{P}^u)$, where S, α are defined as in Definition 1; $\mathbf{P}^l, \mathbf{P}^u : S \times S \to [0, 1]$ are two transition probability matrices, where $\mathbf{P}^l(s, s')$ (resp. $\mathbf{P}^u(s, s')$) gives the *lower* (resp. *upper*) bound of the transition probability from state s to s'.

Semantics. There are two semantic interpretations of IMCs [27], i.e., *Uncertain Markov Chains* (UMC) and *Interval Markov Decision Processes* (IMDP). In this paper, following [7], we mainly focus on the UMC semantics. An IMC $\mathcal{I} = (S, \alpha, \mathbf{P}^l, \mathbf{P}^u)$ represents an infinite set of MCs, denoted by $[\mathcal{I}]$, where for each MC $(S, \alpha, \mathbf{P}) \in [\mathcal{I}]$, $\mathbf{P}^l(s, s') \leq \mathbf{P}(s, s') \leq \mathbf{P}^u(s, s')$ for all pairs of states $s, s' \in S$. Intuitively, under this semantics we assume that the external environment nondeterministically selects an MC from the set $[\mathcal{I}]$ at the beginning and then all the transitions take place according to the chosen MC. Without loss of generality, *we only consider IMC $\mathcal{I}$ with* $[\mathcal{I}] \neq \emptyset$, i.e., there exists at least one implementation. This condition can be easily checked.

Similar to MCs, we can also view an IMC as a digraph such that there is an edge from s to t iff $\mathbf{P}^u(s, t) > 0$. In this way, we can speak of the set of all SCCs and BSCCs of an IMC $\mathcal{I}$ which we denote by $\mathcal{E}(\mathcal{I})$ and $\mathcal{B}(\mathcal{I})$, respectively.

For complexity consideration, for the introduced probabilistic models, we assume that all the probabilities are rational numbers. We define the size of $\mathcal{D}$ (resp. $\mathcal{I}$), denoted by $\sharp\mathcal{D}$ (resp. $\sharp\mathcal{I}$), as the size of the representation of $\mathcal{D}$ (resp. $\mathcal{I}$). Here rational numbers (probabilities) are represented as quotients of integers written in binary. The size of a rational number is the sum of the bit lengths of its numerator and denominator and the size of a matrix is the sum of the sizes of its entries. When stating a complexity result, we assume the standard Turing model.

2.2 Information theory

For a RV X with outcomes $\{x_1, \cdots, x_n\}$, the *Shannon entropy* of X is defined as $\mathbb{H}(X) = -\sum_{i=1}^{n} p(x_i)\log p(x_i)$. (Note that by convention we define $0\log 0 = 0$ as $\lim_{x\to 0} x\log x = 0$). All logarithms are to the base 2; however our results are *independent* of the base. The definition of Shannon entropy is easily generalised to *joint entropy*, the entropy of several RVs computed jointly. Namely $\mathbb{H}(X_1, \cdots, X_n) = -\sum_{x_1\in\mathcal{X}_1}\cdots\sum_{x_n\in\mathcal{X}_n} p(x_1,\cdots,x_n)\log p(x_1,\cdots,x_n)$. We also define *conditional entropy* which quantifies the amount of information needed to describe the outcome of a random variable Y given that the value of another random variable X is known. Namely $\mathbb{H}(Y|X) = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} p(x,y)\log\dfrac{p(x)}{p(x,y)}$. The *chain rule* relates the joint entropy and the conditional entropy, namely, $\mathbb{H}(Y|X) = \mathbb{H}(X,Y) - \mathbb{H}(X)$. It follows that the joint entropy can be calculated using conditional entropy, i.e., $\mathbb{H}(X_0, \cdots, X_n) = \mathbb{H}(X_0) + \mathbb{H}(X_1|X_0) + \cdots + \mathbb{H}(X_n|X_1, \cdots, X_{n-1})$.

3 Entropy of MCs

In this section, we define and characterise the entropy/entropy rate for an MC which we fix to be $\mathcal{D} = (S, \alpha, \mathbf{P})$. $\mathcal{D}$ is equipped with a stochastic process as $\{X_n\}_{n\in\mathbb{N}}$. Let's start from a basic property which can be deduced from the memoryless property.

▶ **Lemma 5.** *$\mathbb{H}(X_n|X_1, \cdots, X_{n-1}) = \mathbb{H}(X_n|X_{n-1})$.*

It turns out that the notion of *local entropy* [7] plays a central role in developing a characterisation of entropy/entropy rate for MCs which are amenable to computation.

▶ **Definition 6** ([7])**.** For any given MC $\mathcal{D}$ and state $s \in S$, the *local entropy* $L(s)$ is defined as $\mathbb{H}(\mathbf{P}(s,\cdot))$, i.e, $-\sum_{t\in S}\mathbf{P}(s,t)\log\mathbf{P}(s,t)$.

3.1 Entropy for absorbing MCs

▶ **Definition 7** ([7])**.** Given an MC $\mathcal{D}$, the entropy of $\mathcal{D}$, denoted $H(\mathcal{D})$, is defined as $H(\mathcal{D}) = \mathbb{H}(X_0, X_1, \cdots)$.

We note that [7] also provides an elegant characterisation. Define $\xi(s) = \sum_{n=0}^{\infty}\pi_s^{(n)}$. (It is called residence time in [7].) Note that basic theory of MCs implies that the state s is *recurrent* if $\xi(s) = \infty$, and is *transient* iff $\xi(s) < \infty$. We write $\vec{\xi}$ for the vector $(\xi(s))_{s\in S}$.

▶ **Theorem 8.** *$H(\mathcal{D}) = \sum_{s\in S} L(s)\xi(s) + \mathbb{H}(\alpha)$, where $\mathbb{H}(\alpha) = -\sum_{s\in S}\alpha(s)\log\alpha(s)$.*

▶ Remark. [7] defines the entropy for general MCs whereas here we assume MCs are absorbing. This does not lose any generality. Mostly we are only interested in MCs with finite entropy, and one easily observes: $H(\mathcal{D})$ is finite *iff* the local entropy of each recurrent state is 0. Note that *absorbing* MCs admits that each recurrent state is made absorbing and thus has local entropy 0.

We also note there is slight difference on $\mathbb{H}(\alpha)$ between our version and that of [7] in Theorem 8. The paper [7] assumes a unique initial state in MCs (i.e., α is Dirac) where $\mathbb{H}(\alpha) = 0$; here we assume a (slightly more) general initial distribution α.

3.2 Entropy rate for general MCs

In contrast to the entropy, the *entropy rate* is defined as

▶ **Definition 9.** Given an MC $\mathcal{D}$, the entropy rate of $\mathcal{D}$, denoted $\nabla H(\mathcal{D})$ is defined as

$$\nabla H(\mathcal{D}) = \lim_{n\to\infty} \frac{1}{n}\mathbb{H}(X_0, \cdots, X_n)$$

As before we characterise $\nabla H(\mathcal{D})$ by local entropy. Define $\zeta(s) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n-1} \pi_s^{(i)}$ and write $\vec{\zeta}$ for the vector $(\zeta(s))_{s\in S}$. We have the following result:

▶ **Theorem 10.** $\nabla H(\mathcal{D}) = \sum_{s\in S} L(s)\zeta(s)$.

▶ Remark. Typically in literature (e.g. [16, 19]), the entropy rate is defined only for an ergodic MC. In that case, one has $\nabla H'(\mathcal{D}) = \lim_{n\to\infty} \mathbb{H}(X_n \mid X_1, \cdots, X_{n-1})$. For ergodic MCs (more generally all stationary processes where MCs are a special case), these two quantities coincide and by Lemma 5, the entropy rate is given by $\nabla H'(\mathcal{D}) = \lim_{n\to\infty} \mathbb{H}(X_n \mid X_{n-1})$.

4 Computing entropy in MCs

In this section, we will focus on the *entropy threshold* problem which asks: given an MC $\mathcal{D}$ and $\theta \in \mathbb{Q}$, does $H(\mathcal{D}) \bowtie \theta$ hold for $\bowtie \in \{\le, <, =, >, \ge\}$? We assume some familiarity with *straight-line programs* and the *counting hierarchy* (cf. [1] or [14]). In particular, the problem *PosSLP* is to decide, given a straight-line program, whether the integer it represents is *positive*. PosSLP belongs to the complexity class $\mathsf{P}^{\mathsf{CH}_3}$ and thus to the fourth-level of the counting hierarchy [1]. We note that counting hierarchy is contained in PSPACE, but it is unlikely to be complete to PSPACE. The following propositions are slight generalisations of [12] and [18], respectively.

▶ **Proposition 11.** *Given $p_1, \cdots, p_n, q_1, \cdots, q_n, \theta \in \mathbb{Q}$, deciding whether $\sum_{i=1}^n p_i \log q_i \bowtie \theta$ for $\bowtie \in \{\le, <, >, \ge\}$ reduces to PosSLP in polynomial time.*

▶ **Proposition 12.** *Given $p_1, \cdots, p_n, q_1, \cdots, q_n, \theta \in \mathbb{Q}$, $\sum_{i=1}^n p_i \log q_i = \theta$ is decidable in polynomial time.*

ABC/Lang-Waldschmidt conjecture implies P. An interesting question is whether one could obtain a lower-bound. This is left as an open question, but the following result somehow discourages such efforts. Indeed, the following proposition can be easily obtained by essentially [18, Proposition 3.7(1)].

▶ **Proposition 13.** *Assume $p_1, \cdots, p_n, q_1, \cdots, q_n, \theta \in \mathbb{Q}$. If the* ABC conjecture *holds, or if the* Lang-Waldschmidt *conjecture holds, then $\sum_{i=1}^n p_i \log q_i \bowtie \theta$ for $\bowtie \in \{\le, <, >, \ge\}$ is decidable in polynomial time.*

Note that the ABC and the Lang-Waldschmidt conjecture (cf. [18] for precise formulations and reference therein) are conjectures in transcendence theory which are widely believed to be true. (For instance, in 2012 there was an announced proof of the ABC conjecture by S. Mochizuki.)

Below we apply these results to the entropy threshold problem of MCs.

4.1 Entropy

Owing to Theorem 8, computing $H(\mathcal{D})$ reduces to computing $\vec{\xi}$. In [7] it is stated that ξ can be computed in polynomial time. Here we need to elaborate this claim to obtain complexity results. This is rather straightforward. For a given absorbing MC which has t transient states and r absorbing states, the transition probability matrix $\mathbf{P}$ can be written as $\mathbf{P} = \begin{bmatrix} Q & R \\ 0 & \mathbf{I}_r \end{bmatrix}$, where Q is a $t \times t$ matrix, R is a nonzero $t \times r$ matrix, and $\mathbf{I}_r$ is an $r \times r$ identity matrix. A basic property of absorbing MCs is that the *fundamental matrix* $\mathbf{I} - Q$ is invertible [21], and we have the following:

▶ **Proposition 14** ([21]). *For absorbing MC, $\vec{\xi} = \alpha'(\mathbf{I} - Q)^{-1}$ where α' is the restriction of α to the t transient states.*

Basic linear algebra reveals that $\vec{\xi}$ can be computed in cubic-time via, e.g., Gauss elimination, and the size of $\vec{\xi}$ is polynomially bounded by $\sharp\mathcal{D}$ (see, e.g., [20]). It then follows from Proposition 11 and Proposition 12 that:

▶ **Theorem 15.** *Given an MC $\mathcal{D}$,*
- *Deciding $H(\mathcal{D}) \bowtie \theta$ for $\bowtie \in \{<, \leq, \geq, >\}$ is in $\mathsf{P}^{\mathsf{CH}_3}$, and is in P assuming the ABC or the Lang-Waldschmidt conjecture.*
- *Deciding $H(\mathcal{D}) = \theta$ is in P.*

4.2 Entropy rate

Owing to Theorem 10, computing $\nabla H(\mathcal{D})$ reduces to computing $\vec{\zeta}$. For (finite) irreducible MC, $\vec{\zeta}$ coincides to the *stationary distribution* π which is unique and independent of the initial distribution. In this case, Theorem 10 yields that $\nabla H(\mathcal{D}) = \sum_{s \in S} L(s)\pi(s)$, which is exactly the classical result, see, e.g., [16]. For general MCs, the transition probability matrix $\mathbf{P}$ has the form

$$\mathbf{P} = \begin{pmatrix} Q & R_1 & R_2 & \cdots & R_h \\ \mathbf{0} & B_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & B_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & B_h \end{pmatrix}$$

where Q corresponds to transient states, and B_i $(1 \leq i \leq h)$ corresponds to the BSCCs (recurrent states).

▶ **Proposition 16.** *For any MC,*

$$\vec{\zeta} = \alpha \cdot \begin{pmatrix} \mathbf{0} & (\mathbf{I} - Q)^{-1} R_1 \mathbf{1}^{\mathrm{T}} \vec{y}_1 & (\mathbf{I} - Q)^{-1} R_2 \mathbf{1}^{\mathrm{T}} \vec{y}_2 & \cdots & (\mathbf{I} - Q)^{-1} R_h \mathbf{1}^{\mathrm{T}} \vec{y}_h \\ \mathbf{0} & \mathbf{1}^{\mathrm{T}} \vec{y}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1}^{\mathrm{T}} \vec{y}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{1}^{\mathrm{T}} \vec{y}_h \end{pmatrix}$$

where $\vec{y}_i$ is the solution of the system of linear equations:

$$\vec{y}_i B_i = \vec{y}_i \text{ and } \mathbf{1}\vec{y} = 1$$

and $\mathbf{1} = (1, \cdots, 1)$.

Similar to the previous section, the size of $\vec{\zeta}$ is polynomially bounded by $\sharp\mathcal{D}$. It then follows from Proposition 11 and Proposition 12 that:

▶ **Theorem 17.** *Given an MC $\mathcal{D}$,*

- *Deciding $\nabla H(\mathcal{D}) \bowtie \theta$ for $\bowtie \in \{<, \leq, \geq, >\}$ is in $\mathsf{P}^{\mathsf{CH}_3}$, and is in P assuming the ABC or the Lang-Waldschmidt conjecture.*
- *Deciding $\nabla H(\mathcal{D}) = \theta$ is in P.*

4.3 Approximation problems

To complete the picture, we show that one can easily approximate $\sum_{i=1}^{n} p_i \log q_i$ up to a given error bound ϵ in polynomial time.

Let $N = n \cdot \max_{1\leq i\leq n} |p_i|$. For each $1 \leq i \leq n$, we can compute $\theta_i \in \mathbb{Q}$ in polynomial-time [10, 18] such that $|\log q_i - \theta_i| < \frac{\epsilon}{N}$ (note that the size of N is bounded polynomially by the size of the input). Observe that

$$\left|\sum_{i=1}^{n} p_i \log q_i - \sum_{i=1}^{n} p_i\theta_i\right| \leq \left|\sum_{i=1}^{n} p_i(\log q_i - \theta_i)\right| \leq \sum_{i=1}^{n} |p_i|\frac{\epsilon}{N} \leq \epsilon.$$

Hence $\sum_{i=1}^{n} p_i\theta_i$, which can be computed in polynomial-time, is an approximation of $\sum_{i=1}^{n} p_i \log q_i$ up to ϵ. Note that, however, unfortunately this does *not* yield an efficient decision procedure for $\sum_{i=1}^{n} p_i \log q_i \bowtie \theta$. It follows that

▶ **Theorem 18.** *Given an MC $\mathcal{D}$ and $\epsilon > 0$, both $H(\mathcal{D})$ and $\nabla H(\mathcal{D})$ can be approximated up to ϵ in polynomial-time in $\sharp\mathcal{D}$ and $\log(\frac{1}{\epsilon})$.*

(Note that this result for entropy is implied in [7] without proof.)

5 Computing the maximum entropy in IMCs

In this section, we turn our attention to IMCs. Recall that an IMC $\mathcal{I}$ represents a set of MCs $[\mathcal{I}]$. We are interested in maximising the entropy/entropy rate of $\mathcal{I}$. The formal definitions are given as follows:

▶ **Definition 19.** Given an IMC $\mathcal{I}$,

- the *maximum entropy* of $\mathcal{I}$, $\overline{H}(\mathcal{I})$, is defined as $\overline{H}(\mathcal{I}) = \sup\{H(\mathcal{D}) \mid \mathcal{D} \in [\mathcal{I}]\}$;
- the *maximum entropy rate* of $\mathcal{I}$, $\overline{\nabla H}(\mathcal{I})$, is defined as $\overline{\nabla H}(\mathcal{I}) = \sup\{\nabla H(\mathcal{D}) \mid \mathcal{D} \in [\mathcal{I}]\}$.

Below we focus on the computation of maximum entropy/entropy rate. In contrast to the previous section, we mainly concentrate on the approximation problem. Results regarding the threshold problem are presented in Section 5.3, though. Throughout this section, we fix an IMC $\mathcal{I} = (S, \alpha, \mathbf{P}^l, \mathbf{P}^u)$.

5.1 Entropy

As pointed out by [7], it could be the case that $\overline{H}(\mathcal{I}) = \infty$ even if for all $\mathcal{D} \in [\mathcal{I}]$, $H(\mathcal{D}) < \infty$. To tackle this issue, an algorithm is given there to determine whether $\overline{H}(\mathcal{I}) = \infty$. In light of this, we *assume that* $\overline{H}(\mathcal{I}) < \infty$. One sufficient condition to guarantee finite maximum entropy is to impose that for any states s and t, $\mathbf{P}^u(s,t) > 0$ implies $\mathbf{P}^l(s,t) > 0$. This is actually a mild assumption in practice (for instance, see [7], Fig. 5). Note that it is also a (lightweight) syntactic way to impose the *Positive* UMC semantics [13].

For $\mathcal{I}$ with $\overline{H}(\mathcal{I}) < \infty$, it cannot be the case that a state is recurrent in some implementation and stochastic in another implementation [7]. Namely, if a state is recurrent in some implementation, it must be deterministic in all implementations, and thus is made absorbing. We denote by $G \subseteq S$ the set of states which are recurrent in *some* implementation of $\mathcal{I}$; G is easily identified by the algorithm in [7].

For each state $s \in S \setminus G$, we introduce a vector of variables $\vec{x}_s = (x_{s,t})_{t\in S}$, and a vector of variables $\vec{y} = (y_s)_{s\in S}$. We define $\Omega(s)$ to be a set of vectors as:

$$\vec{x}_s \in \Omega(s) \ \text{ iff } \ \begin{cases} \sum_{t\in S} x_{s,t} = 1 \\ \mathbf{P}^l(s,t) \le x_{s,t} \le \mathbf{P}^u(s,t), \ \text{for each } t \in S \end{cases} \tag{1}$$

(Note that here we abuse the notation slightly by identifying variables and *valuations* of the variables.) For simplicity, we define, for $\vec{x}_s$ and $\vec{y}$,

$$\Gamma(\vec{x}_s, \vec{y}) = \sum_{t\in S} x_{s,t} y_t - \sum_{t\in S} x_{s,t} \log x_{s,t} \ . \tag{2}$$

We then consider the following non-linear program over $\vec{x}_s$ for all $s \in S \setminus G$ and $\vec{y}$:

$$\begin{array}{lll} \text{minimise} & \displaystyle\sum_{s\in S\setminus G} \alpha(s) y_s & \\ \text{subject to} & y_s \ge \displaystyle\max_{\vec{x}_s\in\Omega(s)} \Gamma(\vec{x}_s, \vec{y}) & s \notin G \\ & y_s = 0 & s \in G \end{array} \tag{3}$$

▶ **Proposition 20.** *The optimal value of* (3) *is equal to* $\overline{H}(\mathcal{I}) - \mathbb{H}(\alpha)$.

We remark that (3) is reminiscent of the *expected total reward* objective (or the stochastic shortest path problem) for MDPs [26, 17, 5]. This does not come in surprise in light of Theorem 8, which might give some intuition underlying (3); cf. [14].

Nevertheless it remains to solve (3). This is rather involved and we only give a rough sketch here. Observe that we have a nested optimisation problem because of the presence of an inner optimisation $\max_{\vec{x}_s\in\Omega(s)} \Gamma(\vec{x}_s, \vec{y})$ in (3). The main strategy is to apply the Lagrange duality to replace it by some "min" (see $\widetilde{\Gamma}$ below). We introduce, apart from $\vec{y}$, variables $\vec{\lambda}^l_s = (\lambda^l_{s,t})_{t\in S}$, $\vec{\lambda}^u_s = (\lambda^u_{s,t})_{t\in S}$ and ν_s for each $s \in S \setminus G$.

It can be shown that (3) is equivalent to

$$\begin{array}{lll} \text{minimise} & \displaystyle\sum_{s\in S\setminus G} \alpha(s) y_s & \\ \text{subject to} & y_s \ge \widetilde{\Gamma}(\vec{\lambda}_s, \nu_s, \vec{y}) & s \notin G \\ & y_s = 0 & s \in G \\ & \lambda^l_{s,t} \ge 0, \lambda^u_{s,t}, \nu_s \ge 0 & s \notin G, t \in S \end{array} \tag{4}$$

where $\widetilde{\Gamma}(\vec{\lambda}_s, \nu_s, \vec{y}) = -\vec{b}_s^{\mathrm{T}}\vec{\lambda}_s^u + \vec{a}_s^{\mathrm{T}}\vec{\lambda}_s^l - \nu_s + e^{-1}\log e \cdot 2^{\nu_s} \cdot (\sum_{t\in S} 2^{\vec{\lambda}_{s,t}^u - \vec{\lambda}_{s,t}^l + y_t})$ and $\vec{a}_s = (\mathbf{P}^l(s,t))_{t\in S}$ and $\vec{b}_s = (\mathbf{P}^u(s,t))_{t\in S}$. (Note that log is to base 2.)

It turns out that (4) is a convex program which can be solved by, e.g., the ellipsoid algorithm or interior-point methods in polynomial time [3, 20]. We obtain

▶ **Theorem 21.** *Given an IMC $\mathcal{I}$ and $\epsilon > 0$, $\overline{H}(\mathcal{I})$ can be approximated upper to ϵ in polynomial-time in $\sharp\mathcal{I}$ and $\log(\frac{1}{\epsilon})$.*

5.2 Entropy rate

In this section, we study the approximation problem for $\overline{\nabla H}(\mathcal{I})$. Firstly we assert that $\overline{\nabla H}(\mathcal{I}) < \infty$ (cf. [14]).

Recall $\mathcal{E}(\mathcal{I})$ is the set of SCCs of $\mathcal{I}$. For each SCC $B \in \mathcal{E}(\mathcal{I})$, we introduce a variable r, a vector of variables $\vec{y} = (y_s)_{s\in B}$, and for each $s \in B$, a vector of variables $\vec{x}_s = (x_{s,t})_{t\in S}$. Recall that $\Omega(s)$ and $\Gamma(\vec{x}_s, \vec{y})$ are defined as in (1) and (2), respectively. We consider the following non-linear program:

$$\begin{array}{lll} \text{minimise} & r & \\ \text{subject to} & r + y_s \geq \max\limits_{\vec{x}_s\in\Omega(s)} \Gamma(\vec{x}_s, \vec{y}) & s \in B \end{array} \tag{5}$$

For each B, we obtain r_B as the optimal value of (5). Note that each state s must belong to a unique $B \in \mathcal{E}(\mathcal{I})$. For simplicity, we define, for a given vector $(z_s)_{s\in S}$, $\Lambda(\vec{x}_s, \vec{z}) = \sum_{t\in S} x_{s,t}\cdot z_t$. We then consider the following non-linear program

$$\begin{array}{lll} \text{minimise} & \sum\limits_{s\in S} \alpha(s) z_s & \\ \text{subject to} & z_s \geq \max\limits_{\vec{x}_s\in\Omega(s)} \Lambda(\vec{x}_s, \vec{z}) & s \in S \\ & z_s \geq r_B & s \in S \text{ and } s \in B \end{array} \tag{6}$$

▶ **Proposition 22.** *$\overline{\nabla H}(\mathcal{I})$ is equal to the optimal value of (6) (which depends on (5)).*

As before, we remark that (6) and (5) are reminiscent of the *limiting average reward* objective for MDPs [26, 5]. This does not come in surprise in light of Theorem 10, which might give some intuition; cf. also [14].

It remains to solve (5) and (6). In the same vein as in Section 5.1, for each B we can *approximate* r_B by some $\theta_B \in \mathbb{Q}$ upper to the given $\epsilon > 0$. We then substitute (6) for each θ_B, and solve the resulting program. It remains to show that (6) does not "propagate" the error introduced in θ_B as it is merely an approximation of the real value r_B. To this end, observe that the optimal value of (6) can be regarded as a function g over $\vec{r} = (r_B)_{B\in\mathcal{E}(\mathcal{I})}$. We have the following result showing the value of (6) is bounded by the "perturbation" of its parameters r_B's. (Note that $\|\cdot\|$ denotes the ∞-norm for vectors.)

▶ **Proposition 23.** *If $\|\vec{r} - \vec{r}'\| \leq \epsilon$, then $|g(\vec{r}) - g(\vec{r}')| \leq \epsilon$.*

We conclude that

▶ **Theorem 24.** *Given an IMC $\mathcal{I}$ and $\epsilon > 0$, $\overline{\nabla H}(\mathcal{I})$ can be approximated upper to ϵ in polynomial-time in $\sharp\mathcal{I}$ and $\log(\frac{1}{\epsilon})$.*

5.3 Threshold problem

In this section, we focus on the maximum entropy/entropy rate *threshold* problem, namely, to decide whether $\overline{H}(\mathcal{I}) \bowtie \theta$ or $\overline{\nabla H}(\mathcal{I}) \bowtie \theta$ for a given $\theta \in \mathbb{Q}$. Recall that we assume $\overline{H}(\mathcal{I}) < \infty$ otherwise the problem is trivial. Below we present two *conditional* decidability results; the unconditional decidability is left as an open problem. We mainly present the results for $\overline{H}(\mathcal{I})$ and the case $\bowtie = \geq$. Other cases can be derived in a similar way and can be found in the full version [14].

By first-order theory. It turns out deciding $\overline{H}(\mathcal{I}) \geq \theta$ amounts to checking

$$\exists \vec{x}, \vec{y}. \bigwedge \begin{cases} \sum_{s \in S \setminus G} \alpha(s) y_s \geq \theta \\ y_s = \sum_{t \in S} x_{s,t} y_t - \sum_{t \in S} x_{s,t} \log x_{s,t} \quad \forall s \in S \setminus G \\ y_s = 0 \quad \forall s \in G \\ \mathbf{P}^l(s,t) \leq x_{s,t} \leq \mathbf{P}^u(s,t) \quad \forall s \in S \setminus G, t \in S \\ \sum_{t \in S} x_{s,t} = 1 \quad \forall s \in S \setminus G \end{cases}$$

where $\vec{x}$ is the concatenation of $\vec{x}_s = (x_{s,t})_{t \in S}$ for $s \in S \setminus G$ and $\vec{y} = (y_s)_{s \in S}$. Recall that G is the set of states which are recurrent in some implementation of $\mathcal{I}$. Evidently this is a formula in the first-order theory of ordered real fields *extended with exponential functions* $(\mathbb{R}, +, -, \cdot, e^x, 0, 1, \leq)$. The theory is known to be o-minimal by the celebrated Wilkie's theorem [29]. However, its decidability is a long-standing open problem in model theory, known as *Tarski's exponential function problem.* A notable result by Macintyre and Wilkie [23] asserts that it is decidable provided the Schanuel's conjecture in transcendence theory is true (which is widely believed to be the case; in fact only a (weaker) real version of the conjecture is needed.) Hence, we obtain a conditional decidability for the maximum entropy threshold problem of IMCs. Note that it is high unlikely that the problem is undecidable, because it would refute the Schanuel's conjecture.

By non-singularity assumption. We can obtain the decidability of the maximum entropy threshold problem by assuming that $\overline{H}(\mathcal{I}) \neq \theta$. To see this, one can simply compute a sequence of approximations of $\overline{H}(\mathcal{I})$ by the approach in Section 5.1, i.e., $\hbar_n$ with $|\overline{H}(\mathcal{I}) - \hbar_n| \leq \frac{1}{2^n}$. The procedure stops when $\hbar_n - \frac{1}{2^n} - \theta$ and $\hbar_n + \frac{1}{2^n} - \theta$ have the same sign. Then $\overline{H}(\mathcal{I}) > \theta$ iff $\hbar_n - \frac{1}{2^n} > \theta$ (or equivalently $\hbar_n + \frac{1}{2^n} > \theta$). Note that we assume $\overline{H}(\mathcal{I}) \neq \theta$, so n must exist as one can take $n = \lceil \log(\frac{1}{|\overline{H}(\mathcal{I}) - \theta|}) \rceil$ although n is not bounded *a priori.*

We conclude this section by the following theorem:

▶ **Theorem 25.** *Given an IMC $\mathcal{I}$. We have that*

- *if the first-order theory of $(\mathbb{R}, +, -, \cdot, e^x, 0, 1, \leq)$ is decidable (which is implied by Schanuel's conjecture), then $\overline{H}(\mathcal{I}) \bowtie \theta$ and $\overline{\nabla H}(\mathcal{I}) \bowtie \theta$ are decidable for $\bowtie \in \{\leq, <, =, >, \geq\}$;*
- *if $\overline{H}(\mathcal{I}) \neq \theta$ (resp. $\overline{\nabla H}(\mathcal{I}) \neq \theta$), then $\overline{H}(\mathcal{I}) \bowtie \theta$ (resp. $\overline{\nabla H}(\mathcal{I}) \bowtie \theta$) is decidable for $\bowtie \in \{\leq, <, >, \geq\}$.*

6 Conclusion

We have studied the complexity of computing (maximum) entropy/entropy rate of Markovian models including MCs, IMCs and MDPs. We obtained a characterisation of entropy rate for general MCs based on which the entropy approximation problem and threshold problem can be solved efficiently assuming number-theoretic conjectures. For IMCs/MDPs, we obtained polynomial-time algorithms to approximate the maximum entropy/entropy rate via convex programming, which improved a result in [7]. We also obtained conditional decidability for the threshold problem.

Open problems include unconditional polynomial-time algorithms for the entropy threshold problem for MCs and unconditional decidability for maximum entropy threshold problem for IMCs/MDPs. Furthermore, we believe it would be promising to explore more algorithmic aspects of information theory along the line of the current work, for instance, for timed automata [2].

References

1 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

2 Nicolas Basset. A maximal entropy stochastic process for a timed automaton,. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 61–73. Springer, 2013.

3 Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications.* Society for Industrial and Applied Mathematics, 1987.

4 Michael Benedikt, Rastislav Lenhardt, and James Worrell. Ltl model checking of interval markov chains. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2013.

5 Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2011.

6 Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wasowski. Quantifying information leakage of randomized protocols. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *VMCAI*, volume 7737 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2013.

7 Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over markov processes. In Adrian Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *LATA*, volume 7810 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2013.

8 Fabrizio Biondi, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. Quail: A quantitative security analyzer for imperative code. In Sharygina and Veith [28], pages 702–707.

9 Michele Boreale. Quantifying information leakage in process calculi. *Inf. Comput.*, 207(6):699–725, 2009.

10 Richard P. Brent. Fast multiple-precision evaluation of elementary functions. *J. ACM*, 23(2):242–251, 1976.

11 Pavol Cerný, Krishnendu Chatterjee, and Thomas A. Henzinger. The complexity of quantitative information flow problems. In *CSF*, pages 205–217. IEEE Computer Society, 2011.

12 Rohit Chadha and Michael Ummels. The complexity of quantitative information flow in recursive programs. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 18 of *LIPIcs*, pages 534–545. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

13 Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval Markov chains. In Roberto M. Amadio, editor, *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008.

14 Taolue Chen and Tingting Han. On the complexity of computing maximum entropy for Markovian models. Technical report, Middlesex University London, 2014. Available via `http://www.cs.mdx.ac.uk/staffpages/taoluechen/pub-papers/fsttcs14-full.pdf`.

15 Taolue Chen, Tingting Han, and Marta Z. Kwiatkowska. On the complexity of model checking interval-valued discrete time markov chains. *Inf. Process. Lett.*, 113(7):210–216, 2013.

16 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley and Sons, Inc., New York, NY, USA, 1991.

17 Luca de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1999.

18 Kousha Etessami, Alistair Stewart, and Mihalis Yannakakis. A note on the complexity of comparing succinctly represented integers, with an application to maximum probability parsing. *TOCT*, 6(2):9, 2014.

19 Valerie Girardin. Entropy maximization for Markov and semi-Markov processes. *Methodology and Computing in Appplied Probability*, 6:109–127, 2004.

20 Martin Grotschel, Laszlo Lovasz, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Algorithms and Combinatorics. Springer-Verlag, 1987.

21 John G. Kemeny and J. Snell. *Finite Markov Chains.* Undergraduate Texts in Mathematics. Springer-Verlag, 3rd printing 1983 edition edition, 1983.

22 Igor Kozine and Lev V. Utkin. Interval-valued finite Markov chains. *Reliable Computing*, 8(2):97–113, 2002.

23 A. J. Macintyre and A. J. Wilkie. On the decidability of the real exponential field. *Odifreddi, P. G., Kreisel 70th Birthday Volume, CLSI*, 1995.

24 William Parry. Intrinsic markov chains. *Trans. Amer. Math. Soc.*, 112:55–66, 1964.

25 Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-time verification of pctl properties of mdps with convex uncertainties. In Sharygina and Veith [28], pages 527–542.

26 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, New York, 1994.

27 Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking Markov chains in the presence of uncertainties. In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006.

28 Natasha Sharygina and Helmut Veith, editors. *Computer Aided Verification – 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science.* Springer, 2013.

29 Alex J. Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted pfaffian functions and the exponential functions. *J. Amer. Math. Soc.*, 9:1051–1094, 1996.

30 Hirotoshi Yasuoka and Tachio Terauchi. Quantitative information flow – verification hardness and possibilities. In *CSF*, pages 15–27. IEEE Computer Society, 2010.

New Time-Space Upperbounds for Directed Reachability in High-genus and H-minor-free Graphs*

Diptarka Chakraborty[1], A. Pavan[2], Raghunath Tewari[3], N. V. Vinodchandran[4], and Lin Forrest Yang[5]

1 **Indian Institute of Technology, Kanpur,** diptarka@cse.iitk.ac.in
2 **Iowa State University,** pavan@cs.iastate.edu
3 **Indian Institute of Technology, Kanpur,** rtewari@cse.iitk.ac.in
4 **University of Nebraska–Lincoln,** vinod@cse.unl.edu
5 **Johns Hopkins University,** lyang@pha.jhu.edu

Abstract

We obtain the following new simultaneous time-space upper bounds for the directed reachability problem. (1) A polynomial-time, $\widetilde{O}(n^{2/3}g^{1/3})$-space algorithm for directed graphs embedded on orientable surfaces of genus g. (2) A polynomial-time, $\widetilde{O}(n^{2/3})$-space algorithm for all H-minor-free graphs given the tree decomposition, and (3) for $K_{3,3}$-free and K_5-free graphs, a polynomial-time, $O(n^{1/2+\epsilon})$-space algorithm, for every $\epsilon > 0$.

For the general directed reachability problem, the best known simultaneous time-space upper bound is the BBRS bound, due to Barnes, Buss, Ruzzo, and Schieber, which achieves a space bound of $O(n/2^{k\sqrt{\log n}})$ with polynomial running time, for any constant k. It is a significant open question to improve this bound for reachability over general directed graphs. Our algorithms beat the BBRS bound for graphs embedded on surfaces of genus $n/2^{\omega(\sqrt{\log n})}$, and for all H-minor-free graphs. This significantly broadens the class of directed graphs for which the BBRS bound can be improved.

1998 ACM Subject Classification F.1.3 [Theory of Computation] Complexity Measures and Classes

Keywords and phrases Reachability, Space complexity, Time-Space Efficient Algorithms, Graphs on Surfaces, Minor Free Graphs, Savitch's Algorithm, BBRS Bound

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.585

1 Introduction

Given a graph G and two vertices s and t, is there a path from s to t in G? This problem, known as *the reachability problem*, is of fundamental importance in the study of space bounded complexity classes as various versions of it characterize important complexity classes (such as NL, RL, L and NC^1 [16, 17, 3]). Progress in understanding the space complexity of graph reachability problems directly relates to the progress in space complexity investigations. We refer the readers to a survey by Wigderson [24] to further understand the significance of reachability problems in complexity theory. Because of its central role, designing space and time efficient deterministic algorithms for reachability problems is a major concern of

* Research supported in part by NSF grants 0916797, 0916525, 1421163, 1422668, and Research-I Foundation.

complexity theory. In this paper we focus on algorithms for reachability over directed graphs that run in polynomial-time and use sub-linear space.

Two basic algorithms for directed reachability are the Breadth First Search algorithm (BFS) and Savitch's algorithm [19]. BFS uses linear space and runs in polynomial time, whereas Savitch's algorithm uses only $O(\log^2 n)$ space, but takes super-polynomial ($\theta(n^{\log n})$) time. Thus BFS is time-efficient and Savitch's algorithm is space-efficient. Hence a natural and significant question that researchers have considered is whether we can design an algorithm for reachability whose time-bound is better than that of Savitch's algorithm and the space-bound is better than that of BFS. A concrete open question is: *Can we design a polynomial-time algorithm for the directed graph reachability problem that uses only $O(n^{1-\epsilon})$ space for some small constant ϵ?* [24].

The best known result in this direction is the bound due to Barnes, Buss, Ruzzo, and Schieber [2]. By cleverly combining BFS and Savitch's algorithm, they designed a polynomial-time algorithm for reachability that uses $O(n/2^{k\sqrt{\log n}})$ space, for any constant k. Henceforth we refer to this bound as the BBRS bound. Improving the BBRS bound remains a significant open question regarding the complexity of the graph reachability problem.

Recently there has been some progress on improving the BBRS bound for certain restricted classes of directed graphs. Asano and Doerr showed that, for any $\epsilon > 0$, there is a polynomial-time algorithm that takes $O(n^{1/2+\epsilon})$ space for reachability over directed *grid graphs* [1]. In [12], it was shown that, for any $\epsilon > 0$, the directed *planar* reachability problem can also be solved in polynomial-time and $O(n^{1/2+\epsilon})$ space. In [20], it was shown that the reachability problem for directed *acyclic* graphs with $O(n^{1-\epsilon})$ *sources nodes* and embedded on surfaces of $O(n^{1-\epsilon})$ genus can be solved in polynomial time and $O(n^{1-\epsilon})$ space. See a recent survey article [22] for more details on known results.

In this paper we design reachability algorithms that beat the BBRS bound for a substantially larger class of graphs than known before. Our main approach is to use a *space-efficient kernelization* where we compress the given graph to a smaller kernel graph preserving reachability. Once such a kernel graph is computed, we can use known algorithms (such as BFS) on the kernel graph to solve the reachability problem.

There are indications that it may be difficult to improve the BBRS bound for general directed graphs using earlier known techniques. This is because there are matching lower bounds known for general reachability on certain restricted model of computation known as NNJAG [5, 15, 9]. All the known algorithms for the general reachability problem can be implemented in NNJAG without significant blow up in time and space. However, we believe that our kernel-based approach has a potential to overcome the NNJAG bottleneck.

Our main motivation to design space-efficient algorithms for reachability problems comes from their importance in computational complexity theory. However, designing polynomial-time, sub-linear space algorithms is of clear significance from a general algorithmic perspective, especially in the context of computations over large data sets. Thus our algorithms may be of interest to a more general audience.

Our Contributions

Our first result is a new algorithm for the directed reachability problem for surface-embedded graphs.

▶ **Theorem 1.** *There is an algorithm that, given a directed graph G embedded on an orientable surface of genus g with the combinatorial embedding and two vertices s and t, decides whether there is a directed path from s to t in G. This algorithm runs in polynomial-time and uses $\tilde{O}(n^{2/3}g^{1/3})$ space, where n is the number of vertices of the graph.*

For the case when $g = n^{1-\epsilon}$, our algorithm uses $\tilde{O}(n^{1-\epsilon/3})$ space and runs in polynomial time (by $\widetilde{O}(s(n))$ we mean $O(s(n)(\log n)^{O(1)})$). In general, for graphs that are embedded on surfaces of genus $g = n/2^{\omega(\sqrt{\log n})}$, our algorithm beats the BBRS bound.

For proving the above theorem, we first give an algorithm for constructing a *planarizing set* (a set S of nodes of a graph G, so that $G \setminus S$ is a planar graph) of size $O(n^{2/3}g^{1/3})$ of the underlying undirected graph in polynomial-time and space $\tilde{O}(n^{2/3}g^{1/3})$. This space-efficient algorithm for computing a planarizing set may be of independent interest.

There are known algorithms that compute a planarizing set of a high-genus graph [8, 11, 10]. However, we cannot rely on these existing algorithms since the starting point of all these algorithms is a BFS tree computation of the input graph. In general computing a BFS tree (even for an undirected graph) is as difficult as the directed reachability problem. Avoiding a BFS tree computation of the entire graph is the the main technical challenge that we overcome in our space efficient algorithm for constructing a planarizing set.

Once a planarizing set is computed, we construct a new directed graph $\tilde{G}$, called the kernel graph on G whose vertex set is the planarizing set, so that reachability in G reduces to reachability in $\tilde{G}$. This reduction uses the $O(n^{1/2+\epsilon})$ space algorithm for directed planar reachability from [12] as a subroutine. Finally we solve reachability on $\tilde{G}$ using BFS. Since the size of $\tilde{G}$ is $O(n^{2/3}g^{1/3})$, we get the desired space bound.

Our second contribution is a new reachability algorithm for H-minor-free graphs, that improves upon the BBRS bound, where H is an arbitrary but fixed graph. To design this algorithm we assume that we are provided with the *tree decomposition* of the H-minor-free graph.

▶ **Theorem 2.** *Given a graph H, there is an algorithm that, given any H-minor-free graph G together with*

(i) *a tree decomposition (T, X) of G, and*

(ii) *for every $X_i \in X$, the combinatorial embedding of the subgraph G_0 of $G[X_i]$,*

and two vertices s and t in G, decides whether there is a directed path from s to t in G. The algorithm runs in polynomial-time and uses $\tilde{O}(n^{2/3})$ space, where n is the number of vertices of the graph.

The reader may refer to Section 4.1 to understand the notation that we use in Theorem 2. This theorem is proved by first designing a $\tilde{O}(n^{2/3})$-space and polynomial-time algorithm for constructing a 2/3-separator of size $O(n^{2/3})$ for the given graph. Once such a separator is obtained, we use ideas from [12] to design the reachability algorithm. To construct such a separator for H-minor-free graphs, we use the tree decomposition of the given graph by [18] and find a "separating node" in that tree. Then we construct a bounded-genus graph from the graph induced by the separating node. Finally by using the planarizing set construction used to prove Theorem 1, we design an algorithm to construct a planarizing set of size $O(n^{2/3})$ of the underlying undirected graph in polynomial-time and $\tilde{O}(n^{2/3})$ space.

For $K_{3,3}$-free and K_5-free graphs we give a better upper bound than the one given in Theorem 2. Kuratowski's theorem states that planar graphs are exactly those graphs that do not contain $K_{3,3}$ and K_5 as minors. Hence it is a natural question whether results on planar graphs can be extended to graphs that do not contain either a $K_{3,3}$ minor (known as $K_{3,3}$-free graphs) or a K_5 minor (known as K_5-free graphs). Certain complexity upper bounds that hold for planar graphs have been shown to hold for $K_{3,3}$-free and K_5-free graphs [4, 21, 6, 7]. On the other hand, there are problems for which upper bounds that hold for planar graphs are not known to extend to such minor-free graphs (such as computing a perfect matching in bipartite graphs [13]). We show that the time-space bound known for planar graphs can also be obtained for both these classes of graphs. Here it is important to note that even though directed reachability in $K_{3,3}$-free and K_5-free graphs reduces to directed planar reachability[21], the reduction blows up the size of the graph by a polynomial factor and hence we can not use this approach for our purposes.

▶ **Theorem 3.** *For any constant $0 < \epsilon < 1/2$, there is a polynomial time and $O(n^{1/2+\epsilon})$ space algorithm that given a directed $K_{3,3}$-free or K_5-free graph G on n vertices, decides whether there is a directed path from s to t in G.*

Although for Theorem 2 we require additional inputs (such as the tree decomposition and the embeddings of the bounded genus parts), in Theorem 3 we do not have any such requirements. The proof idea of Theorem 3 is similar to that of Theorem 2. However we use the known algorithm to compute a planar separator instead of a bounded genus separator. This gives better space bound compared to the case of H-minor-free graphs.

The rest of the paper is organized as follows. In Section 2 we give some basic definitions and notations that we use. In Section 3, we give a construction of planarizing set for high-genus graphs and also provide a proof of Theorem 1. In Section 4, we present the algorithm for reachability in H-minor-free graphs and as a corollary we show Theorem 3. Due to space constraints, most of the proofs appear in the Appendix.

2 Preliminaries

We first define some notations which will be used later in this paper. Given a graph G and a set of vertices X, $G[X]$ denotes the subgraph of G induced by X and $V(G)$ denotes the set of vertices present in the graph G. Now we define necessary notions on graphs embedded on surfaces. We refer the reader to the excellent book by Mohar and Thomassen [14] for a comprehensive treatment of this topic. In this paper we only consider *closed orientable* surfaces. These surfaces are obtained by adding "handles" to a sphere.

Let $G = (V, E)$ be a graph and for each $v \in V$, let π_v be a cyclic permutation of edges incident on v. Let $\Pi = \{\pi_v \mid v \in V\}$. We say that Π is a *combinatorial embedding* of G. Given a combinatorial embedding we can define Π-*facial walk*. Let $e = \langle v_1 v_2 \rangle$ be an edge. Consider the closed[1] walk $f = v_1 e_1 v_2 e_2 v_3 \cdots v_k e_k v_1$ where $\pi_{v_{i+1}}(e_i) = e_{i+1}$, and $\pi_{v_1}(e_k) = e_1$. We call f a *face* of the graph G.

Given a Π-embedding of a graph G, the Π-*genus* of G is the g such that $n - e + f = 2 - 2g$, where n, e and f denote the number of vertices, edges and faces of the graph G. This is popularly known as the *Euler-Poincaré formula.*

It is known that given any graph with Π-genus g, it can be embedded on a closed orientable surface of genus g such that every face is homeomorphic to an open disc. Let Π be a combinatorial embedding of a graph G and H be a subgraph of G. The embedding Π naturally induces an embedding Π' on $G \setminus H$. By abuse of notation, we still refer to the induced embedding as Π-embedding.

Given a cycle C of a graph, we can define *left* and *right* sides of the cycle C. Two vertices are on the same side of C if they are path connected such that the path does not cross the cycle C. We use $G_l(C)$ and $G_r(C)$ to denote the left and right sides of G. Given a cycle C, we say that it is *contractible* if one of $G_l(C) \cup C$ or $G_r(C) \cup C$ has Π-genus zero (i.e. planar). We say that a cycle is *surface separating* if $G_l(C)$ and $G_r(C)$ have no edges in common. Note

[1] A priori it is not obvious that that this leads to a closed walk. However, it can shown that this walk comes back to v_1. See [14] Chapter 3.2 for a proof.

that every contractible cycle is surface separating. A cycle that is not surface separating is called a *non-separating cycle*. We now mention some fundamental facts about these cycles that are used throughout this paper.

▶ **Proposition 1.** *Let C be a cycle of a graph with Π-genus g. If C is non-separating, then Π-genus of $G \setminus C$ is $\leq g-1$. If C is surface separating, then sum of Π-genera of $G_l(C) \cup C$ and $G_l \cup C$ equals g.*

An edge that appears on a facial walk f may appear once or twice on f. Any edge that appears twice on a facial walk is called *singular edge*.

▶ **Proposition 2.** *Let G be a graph with Π-genus g, and e be a singular edge such that $G \setminus e$ is connected. The Π-genus of $G \setminus e$ is $g-1$.*

The notions of planarizing set and separator defined below are crucial in this paper. A set S of vertices of a graph G is called a *planarizing set* if $G \setminus S$ is a planar graph. An (α, β)-separator of a graph $G = (V, E)$ having n vertices, is a subset S of V such that $|S| \leq O(\alpha)$ and every connected component in $V \setminus S$ has at most βn vertices.

Next we state two theorems about planar graphs that are used in this paper. In [12] the authors construct a $(n^{1/2}, 8/9)$-separator. By running their algorithm repeatedly (a constant number of times), we can obtain a $(n^{1/2}, 1/3)$ separator.

▶ **Theorem 4** ([12])**.** *Given a planar graph G there is an algorithm that computes a $(n^{1/2}, 1/3)$-separator of G in polynomial time and $\tilde{O}(n^{1/2})$ space.*

We refer to the algorithm of this theorem as `PlanarSeparator` algorithm. In [12], this algorithm is used to obtain a time-space efficient algorithm for reachability on directed planar graphs

▶ **Theorem 5** ([12])**.** *For any constant $0 < \epsilon < 1/2$, there is an algorithm that, given a directed planar graph G and two vertices s and t, decides whether there is a path from s to t. This algorithm runs in time $n^{O(1/\epsilon)}$ and uses $O(n^{1/2+\epsilon})$ space, where n is the number of vertices of G.*

3 A Reachability Algorithm for High Genus Graphs

In this section we prove Theorem 1. We will use a space-efficient construction of a planarizing set to establish this result. We first assume that the following theorem holds and then prove Theorem 1. Proof of Theorem 6 will appear in Section 3.1.

▶ **Theorem 6.** *There is an algorithm that given a combinatorial embedding of an undirected graph G embedded on an orientable surface of genus g, outputs a planarizing set of G of size $O(n^{2/3}g^{1/3})$. This algorithm runs in polynomial time and uses space $\tilde{O}(n^{2/3}g^{1/3})$. Here n denotes the number of vertices of G.*

Proof of Theorem 1. Let $\langle G, s, t\rangle$ be an instance of reachability where G whose Π-genus is g. Consider the underlying undirected graph G_{un}. By using the algorithm from Theorem 6 we first compute a planarizing set S of G_{un}. Let $\mathcal{S} = S \cup \{s, t\}$. Let G_p be the planar graph obtained by removing all vertices (and the edges incident on them) of $\mathcal{S}$ from G.

Consider the following reduction that outputs an instance $\langle \mathcal{G}, s, t\rangle$, where $\mathcal{G} = (\mathcal{S}, \mathcal{E})$. Given two nodes a and b in $\mathcal{S}$, we place a directed edge from a to b in $\mathcal{E}$, if there is a directed edge from a to b in the original directed graph G. Additionally, we place an edge from a to

b in $\mathcal{E}$, if there exist vertices u and v in the vertex set of G_p such that all of the following conditions hold: 1) there is a directed edge from a to u in G, 2) there is a directed edge from v to b in G, and 3) there is a directed path from u to v in the directed planar graph G_p. Determining whether there is path from u to v in G_p can be done in polynomial-time and $O(n^{2/3})$ space, by setting ϵ to 1/6 by Theorem 5. By Theorem 6, S can be computed in polynomial time and $\tilde{O}(n^{2/3}g^{1/3})$ space. Thus this reduction runs in polynomial time and uses $\tilde{O}(n^{2/3}g^{1/3})$ space.

We now claim that there is a path from s to t in G if and only if there is a path from s to t in $\mathcal{G}$. Consider any s-t path in $\mathcal{G}$, let $e_1, e_2, \cdots e_k$ be the edges of this path. Consider an edge $e_i = (a, b)$. Note that the reduction places this edge in $\mathcal{G}$ when, either there is a directed path or an edge from a to b in G. This implies that there is path from s to t in G. Now we prove the converse direction. Let P be a path from s to t in G. We can decompose P into $p_1e_1q_1h_1p_2e_2q_2h_2 \cdots p_k$. Here e_i is an edge from a vertex in $\mathcal{S}$ to a vertex in G_p and h_i is an edge from a vertex in G_p to a vertex in $\mathcal{S}$, q_i is the part of the path P from head of e_i to the tail of h_i so that it completely lies within G_p, and p_i is the part of the path P that completely lies in the graph induced by the planarizing set $\mathcal{S}$. By the construction of $\mathcal{G}$, there is an edge o_i from the tail of e_i to the head of h_i in $\mathcal{G}$. Thus $p_1o_1p_2o_2 \cdots p_k$ is a path from s to t in $\mathcal{G}$.

Reachability in the directed graph $\mathcal{G}$ can be solved using BFS. Since the number of vertices in $\mathcal{G}$ is $O(n^{2/3}g^{1/3})$, the BFS algorithm runs in polynomial-time and uses in $\tilde{O}(n^{2/3}g^{1/3})$ space. By combining the above reduction with the reachability algorithm on $\mathcal{G}$, we obtain an algorithm that solves reachability in G that runs in polynomial time and uses $\tilde{O}(n^{2/3}g^{1/3})$ space. This completes the proof of Theorem 1. ◀

3.1 Proof of Theorem 6

The structure of the proof is as follows. Given an embedded graph, we decompose the graph into several regions. We first look for a small non-contractible cycle C inside some region. If we find one, then we add the vertices of C into the planarizing set. If C is non-separating, by Proposition 1, removal of the vertices of C will result in a graph whose genus $\leq g-1$. If C is surface separating, since C is non-contractible, by Proposition 1, we get two components each with genera $0 < g_1, g_2 < g$ so that $g_1 + g_2 = g$. In both cases, since the genus of each component is $< g$, we can iterate this process. If this iteration stops, then all the regions of all the resulting components are homeomorphic to an open disc. In this case, for each component we identify a small subgraph based on the regions, and argue that this subgraph is a planarizing set of that component. Our final planarizing set is the collection of planarizing sets of each component together with the non-contractible cycles. Notice that at any stage the components obtained can be implicitly represented by the original graph and the cycles that are removed. Thus we do not have to explicitly store the components. We only store the non-contractible cycles that are removed. We now proceed to give a formal proof. The algorithm given in the following lemma is the core of the planarizing set algorithm.

▶ **Lemma 7.** *There is an algorithm that given a connected undirected graph G, its Π-embedding, and an integer k as input, outputs one of the following:*

1. *A non-separating cycle of size $O(k)$ or a singular edge e so that $G \setminus e$ is connected. The output of this step (either a cycle or a singular edge) is called a genus reduction set.*
2. *a non-contractible and surface-separating cycle of size $O(k)$*
3. *a planarizing set of size $O((n/k + g)\sqrt{k})$*

The algorithm runs in polynomial-time and uses $\tilde{O}(n/k + k)$ space.

The proof of the above lemma is given in the Appendix. Now using this lemma, we prove Theorem 6.

Proof of Theorem 6. The planarizing set construction algorithm applies the algorithm from Lemma 7 iteratively. We will describe the algorithm by describing an iteration. After the i^{th} iteration, we will have a collection of components $G_1, G_2, \ldots, G_m$. We will describe the $(i+1)^{st}$ iteration: The algorithm considers the first component $\hat{G}$ whose Π-genus $\hat{g}$ is non-zero and apply the algorithm from Lemma 7 on $\hat{G}$. This results in either (1) a genus-reduction set of $\hat{G}$, (2) non-contractible surface separating cycle of $\hat{G}$, or (3) planarizing set of $\hat{G}$. In cases (1) and (2) the algorithm stores the corresponding cycles. In case (3) it adds the planarizing set obtained to the final planarizing set. This process stops when all the components are planar.

We claim that after any iteration, the total number of vertices in all of the components together is at most n, and the total genera of all of the components together is at most g. Assume that this claim holds after i^{th} iteration. Let $\hat{G}$ be the component considered at the $(i+1)^{st}$ iteration. In case (1), by Propositions 1 and 2, $\hat{G}$ is reduced to a component whose genus is at most $\hat{g}-1$. In case (2), since we have a non-contractible surface separating cycle, by Proposition 1, we get two components whose sum of the genera is at most $\hat{g}$. In case (3), $\hat{G}$ is reduced to a planar graph. Thus sums of the genera of all components is $\leq g$ and, since no vertex is repeated in more than one component, vertices in all of the components together is at most n.

Clearly this algorithm produces a planarizing set and runs in polynomial-time. We will now bound the size of the planarizing set and the space used by the algorithm.

Notice that the algorithm stores only the cycles and singular edges and will not store the components: At any stage, given the original graph, the cycles or singular edges computed so far, and an index of the component, the edge relations of that component can be computed without additional space. After at most g iterations, we are left with at most g components each of whose genus is at most 1. Since each iteration may produce a cycle of length $O(k)$, the algorithm will store at most $2g$ cycles each of length $O(k)$. Consider a component G_i in which case (3) of the lemma happens. The size of the corresponding planarizing set produced is $O(n_i/k + g_i)\sqrt{k}$. Since $\sum_i n_i \leq n$ and $\sum_i g_i \leq g$, the total size of the planarizing set is $O((n/k+g)\sqrt{k}+kg)$. Total space used is $\tilde{O}(n/k+k+kg+(n/k+g)\sqrt{k})$ (including the space to store the planarizing set).

By choosing $k = \max\{(n/g)^{2/3}, 1\}$, we get that the total space-bound of the algorithm to compute the planarizing set is $\tilde{O}(n^{2/3}g^{1/3})$, and the size of the planarizing set produced is $O(n^{2/3}g^{1/3})$. ◀

4 A Reachability Algorithm for H-minor-free Graphs

In this section, we prove Theorem 2 by first giving an algorithm to construct a separator of the input graph. Towards this we define the notion of a tree decomposition of a graph which is crucial to the construction.

4.1 Graph Minor Decomposition Theorem

A graph H is said to be a *minor* of a graph G if H can be obtained from a subgraph of G by contracting some edges. A graph G is said to be *H-minor-free* if G does not contain H as a minor, for some graph H.

▶ **Definition 8.** A *tree decomposition* of a graph $G = (V, E)$ is the tuple (T, X) where $T = (V_T, E_T)$ is a tree and $X = \{X_i \mid i \in V_T\}$, such that, (a) $\cup_i X_i = V$, (b) for every edge (u, v) in G, there exists i, such that u and v belong to X_i, and (c) for every $v \in V$, the set of nodes $\{i \in V_T \mid v \in X_i\}$ forms a connected subtree of T.

We will refer to the X_i's as *bags* of vertices. Note that each bag corresponds to a node (we call vertices of T as *nodes*) in the tree T. The *width* of a tree decomposition (T, X), is the maximum over the size of X_i's minus 1. The *treewidth* of a graph is the minimum width over all possible tree decompositions of G. A tree decomposition is said to be a path decomposition if $T = (V_T, E_T)$ is a path and *pathwidth* of a graph is the minimum width over all possible path decompositions of G.

For a fixed graph H, Robertson and Seymour, gave a tree decomposition for every H-minor-free graph [18]. Before we see the Theorem we need to state some definitions.

A graph G is called *almost h-embeddable* if there exists a set of vertices Y (called the *apices*) of size at most h such that, (i) $G \setminus Y$ can be written as $G_0 \cup G_1 \cup \ldots \cup G_h$, (ii) G_0 has an embedding on a surface of genus at most h (say $\mathcal{S}$), (iii) for $i = 1, \cdots, h$, G_i's are pairwise disjoint (we shall refer to them as *vortices*), (iv) there exists faces $F_1, \cdots, F_h$ of G_0 and pairwise disjoint discs $D_1, \cdots, D_h$ on $\mathcal{S}$ such that for all $i \in \{1, \ldots, h\}$, $D_i \subseteq F_i$ and $U_i := V(G_0) \cap V(G_i) = V(G_0) \cap D_i$, and (v) for each graph G_i, there is a path decomposition $(\mathcal{P}_u)_{u \in U_i}$ of width at most h such that $u \in \mathcal{P}_u$, for all $u \in U_i$. The sets of vertices in $\mathcal{P}_u$ are ordered according to the ordering of the corresponding u's as vertices along the boundary of face F_i in G_0.

Let G and H be two graphs each containing cliques of equal sizes. The *clique-sum* of G and H is formed by identifying pairs of vertices in these two cliques to form a single shared clique, and then possibly deleting some of the clique edges (may be none). A *k-clique-sum* is a clique-sum in which both cliques have at most k vertices. The k-clique-sum of G and H is denoted as $G \oplus_k H$. The set of shared vertices in this operation is called the *join set.*

We are now ready to state the decomposition theorem for H-minor-free graphs.

▶ **Theorem 9** ([18]). *For every graph H, depending only on $|V(H)|$, there exists an integer $h \geq 0$ such that every H-minor-free graph can be represented as at most h-clique-sum of "almost h-embeddable" graphs in some surface on which H cannot be embedded.*

Henceforth, we will assume that the tree decomposition of the original graph and the combinatorial embedding of all subgraphs (the G_0's in each almost h-embeddable graph) that are embedded on the surface are provided as part of the input. We will refer to this as *tree decomposition with combinatorial embedding* of H-minor-free graphs.

4.2 Constructing Separator for H-minor-free Graphs

Now we will show that given a decomposition of a H-minor-free graph stated in the last subsection, how to construct a separator. We start with the following lemma whose proof is given in the Appendix.

▶ **Lemma 10.** *There exists a log-space algorithm, that given a tree decomposition (T, X) of a graph G on n vertices, outputs a node $i \in T$ such that every connected component in $G[V \setminus X_i]$ has at most $n/2$ vertices.*

We now give a separator construction for all H-minor-free graphs which is the main contribution of this whole section.

▶ **Theorem 11.** *Given a H-minor-free graph G and its tree decomposition with combinatorial embedding, there exists an $\tilde{O}(n^{2/3})$ space, polynomial time algorithm that computes a $(n^{2/3}, 2/3)$-separator of G.*

Proof. Given an input graph G and its tree decomposition, compute the vertex i using Lemma 10. The separator for G that we would construct would be a subset of X_i. Let i have m neighbors in T, say $i_1, \ldots, i_m$. Now for every $j \in [m]$, $G[X_i]$ is joined with $G[X_{i_j}]$ using the clique-sum operation of at most h (constant depending only on H) vertices. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ where C_j is a set of at most h vertices in X_i, such that $G[X_i]$ is joined with $G[X_{i_j}]$ via C_j. Let T_j be the connected subtree of $T \setminus i$ containing the node j. We define the subgraph G_j to be the induced subgraph of G corresponding to the vertices in the subtree T_j. In other words, $G_j = G[\cup_{l \in T_j} X_l]$. Let $k_j = |G_j|$.

Now if $|X_i| \le O(n^{2/3})$, then it follows from Lemma 10 that X_i is a $(n^{2/3}, 1/2)$-separator of G. Otherwise, consider the node i and its corresponding almost h-embeddable graph $K = G[X_i]$. Now consider the representation of K using apices and vortices. Let Y be the set of apices and $K \setminus Y$ can be written as $K_0 \cup K_1 \cup \cdots \cup K_h$ where each of K_i has a path decomposition $(\mathcal{P}_u)_{u \in U_i}$ of width less than h. Now build a new graph K' from K_0 using the following steps: for $i = 1, \cdots, h$, add a cycle of length $|\mathcal{P}_u|$ attached to the vertex $u \in U_i$ inside the face F_i and then connect those cycles such that they form a path like structure similar to the corresponding path decomposition. The new graph K' is a graph embedded on a constant genus and so from Theorem 6, we can get a $(n^{2/3}, 2/3)$-separator S (which is union of planarizing set of K', say Z and output of `PlanarSeparator` on the graph $K' \setminus Z$) using $\tilde{O}(n^{2/3})$ space and polynomial time. If S contains some vertices from a newly added cycle, then we add all the vertices present in the corresponding "bag" of vertices of the respective path decomposition. We also add all the apices of K_0 and we get a new set S'. As the size of S is $O(n^{2/3})$, so the size of S' will be at most $O(hn^{2/3}) = O(n^{2/3})$.

▶ **Claim 1.** *S' is a $(n^{2/3}, 2/3)$-separator of K.*

Proof. Observe that by construction, K' is a graph embedded on a bounded genus surface. Moreover there is a canonical injective map (say σ) from vertices in K to vertices in K'. To see this, note that $K' = K_0 \cup$ {newly added cycles} and by construction, for every vertex in the bag X_i there is a vertex in the newly added cycle in K'.
Since S is a $(n^{2/3}, 2/3)$-separator of K', S' is also a $(n^{2/3}, 2/3)$-separator of K. Let C be a connected component in $K \setminus S'$. Then the vertices corresponding to C in K' (via the map σ) also form a connected component. Since every connected component in $K' \setminus S$ has size at most $2|K'|/3$, so S' is a $(n^{2/3}, 2/3)$-separator of K. ◀

By running the above construction repeatedly (a constant number of times), we can get a $(n^{2/3}, 1/6)$-separator $\overline{S}$. As according to Lemma 10, $G[V \setminus X_i]$ contains at most $n/2$ vertices, so the set $\overline{S}$ also acts as a $(n^{2/3}, 2/3)$-separator for the whole graph G. It is clear from the construction of $\overline{S}$ that this algorithm will take $\tilde{O}(n^{2/3})$ space and polynomial time. ◀

We also consider the special case when H is either the $K_{3,3}$ or the K_5.

▶ **Theorem 12** ([23, 21])**.** *Let (T, X) be a tree decomposition of a $K_{3,3}$-free or K_5-free graph G. Then*
(i) *for every $X_i \in X$, $G[X_i]$ is either a planar graph or the K_5 (if G is $K_{3,3}$-free) or V_8 (if G is K_5-free), and*
(ii) *G is the 3-clique-sum of $G[X_i]$ and $G[X_j]$ for every adjacent vertices i, j in T.*

Moreover given a $K_{3,3}$-free or K_5-free graph G, such a tree decomposition can be computed in logspace.

Thierauf and Wagner have shown how to compute the tree decomposition of a $K_{3,3}$-free or K_5-free graph given in Theorem 12 in log-space [21] and thus we get the following corollary for these special class of H-minor-free graphs.

▶ **Corollary 13.** *Given a $K_{3,3}$-free or K_5-free graph G, there exists an $\tilde{O}(n^{1/2})$ space, polynomial time algorithm that computes a $(n^{1/2}, 2/3)$-separator of G.*

The detailed proof of the above stated corollary is given in the Appendix.

Proof of Theorem 2. Observe that the planar reachability algorithm of Theorem 5 essentially uses the properties that (I) a subgraph of a planar graph is also planar, and (II) their exists an algorithm that computes a $(n^{1/2}, 2/3)$-separator of a planar graph in polynomial time and $\tilde{O}(n^{1/2})$ space. Note that by the definition itself, all the subgraphs of a H-minor-free graph is also H-minor-free and given a tree decomposition, from Theorem 11 we get an algorithm that computes a $(n^{2/3}, 2/3)$-separator of a H-minor-free graph in polynomial time and $\tilde{O}(n^{2/3})$ space. Now using the algorithm stated in Theorem 5, we get our desired result. ◀

Just mimicking the above proof, we can achieve a better simultaneous time-space bound for the directed reachability problem over $K_{3,3}$-free or K_5-free graphs as stated in Theorem 3 using the separator obtained from the Corollary 13.

References

1 Tetsuo Asano and Benjamin Doerr. Memory-constrained algorithms for shortest path problem. In *CCCG*, 2011.

2 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, pages 27–33, 1992.

3 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

4 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009.

5 Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.*, 9(3):636–652, 1980.

6 Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. In *Annual IEEE Conference on Computational Complexity*, pages 203–214, 2009.

7 Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Graph isomorphism for $K_{\{}3,3\}$-free and K_5-free graphs is in log-space. In *FSTTCS*, pages 145–156, 2009.

8 Hristo Djidjev and Shankar M. Venkatesan. Planarization of graphs embedded on surfaces. In *Workshop on Graph Theoretic Concepts in Computer Science*, pages 62–72, 1995.

9 Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for st-connectivity on the NNJAG model. *SIAM J. Comput.*, 28(6), 1999.

10 John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5(3):391–407, 1984.

11 Joan P. Hutchinson and Gary L. Miller. Deleting vertices to make graphs of positive genus planar. *Discrete Algorithms and Complexity Theory*, 1986.

12 T. Imai, K. Nakagawa, A. Pavan, N. V. Vinodchandran, and O. Watanabe. An $O(n^{1/2+\epsilon})$-Space and Polynomial-Time Algorithm for Directed Planar Reachability. In *IEEE Conference on Computational Complexity (CCC)*, pages 277–286, 2013.

13 Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24:1002–1017, 1995.

14 Bojan Mohar and Carsten Thomassen. Graphs on surfaces. 2001. *Johns Hopkins Stud. Math. Sci*, 2001.

15 Chung Keung Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *FOCS*, pages 218–227, 1993.

16 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008.

17 Omer Reingold, Luca Trevisan, and Salil Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *STOC'06: Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*, pages 457–466, New York, NY, USA, 2006. ACM.

18 Neil Robertson and P. D. Seymour. Graph minors. xvi. excluding a non-planar graph. *J. Comb. Theory Ser. B*, 89(1):43–76, September 2003.

19 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4:177–192, 1970.

20 Derrick Stolee and N. V. Vinodchandran. Space-efficient algorithms for reachability in surface-embedded graphs. In *IEEE Conference on Computational Complexity (CCC)*, pages 326–333, 2012.

21 Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free Graphs and K_5-free Graphs is in Unambiguous Log-Space. In *17th International Conference on Foundations of Computation Theory (FCT)*, Lecture Notes in Computer Science 5699, pages 323–334. Springer-Verlag, 2009.

22 N. V. Vinodchandran. Space complexity of the directed reachability problem over surface-embedded graphs. Technical Report TR14-008, ECCC, 2014.

23 K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937.

24 Avi Wigderson. The complexity of graph connectivity. *Mathematical Foundations of Computer Science 1992*, pages 112–132, 1992.

Polynomial Min/Max-weighted Reachability is in Unambiguous Log-space

Anant Dhayal, Jayalal Sarma, and Saurabh Sawlani

Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai, India

Abstract

For a graph $G(V, E)$ and a vertex $s \in V$, a weighting scheme ($w : E \to \mathbb{N}$) is called a *min-unique* (resp. *max-unique*) weighting scheme, if for any vertex v of the graph G, there is a unique path of minimum(resp. maximum) weight[1] from s to v. Instead, if the number of paths of minimum(resp. maximum) weight is bounded by n^c for some constant c, then the weighting scheme is called a *min-poly* (resp. *max-poly*) weighting scheme.

In this paper, we propose an unambiguous non-deterministic log-space (UL) algorithm for the problem of testing reachability in layered directed acyclic graphs (DAGs) augmented with a *min-poly* weighting scheme. This improves the result due to Reinhardt and Allender [11] where a UL algorithm was given for the case when the weighting scheme is *min-unique*.

Our main technique is a triple inductive counting, which generalizes the techniques of [7, 12] and [11], combined with a hashing technique due to [5] (also used in [6]). We combine this with a complementary unambiguous verification method, to give the desired UL algorithm.

At the other end of the spectrum, we propose a UL algorithm for testing reachability in layered DAGs augmented with *max-poly* weighting schemes. To achieve this, we first reduce reachability in DAGs to the longest path problem for DAGs with a unique source, such that the reduction also preserves the *max-poly* property of the graph. Using our techniques, we generalize the double inductive counting method in [8] where UL algorithms were given for the longest path problem on DAGs with a unique sink and augmented with a *max-unique* weighting scheme.

An important consequence of our results is that, to show NL = UL, it suffices to design log-space computable *min-poly* (or *max-poly*) weighting schemes for DAGs.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Alternation and Non-determinism

Keywords and phrases Reachability Problem, Space Complexity, Unambiguous Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.597

1 Introduction

Reachability testing in graphs (REACH) is an important algorithmic problem that encapsulates central questions in space complexity. Given a graph $G(V, E)$ and two special vertices s and t, the problem asks to test if there is a path from s to t in the graph G. The problem admits a (deterministic) log-space algorithm for the case of trees and undirected graphs (by a breakthrough result due to Reingold[10]). The directed graph version of the problem captures the complexity class NL. Designing a log-space algorithm for the problem is equivalent to proving NL = L. (See [1] for a survey.) Even in the case when the graph is a layered DAG[2], the problem is known to be NL-complete.

[1] Weight of a path p is the sum of the weights of the edges appearing in p.
[2] A DAG is layered, if V can be partitioned as $V = V_1 \cup \ldots V_\ell$ s.t. edges go from V_i to V_{i+1} for some i.

An important intermediate class of algorithms for reachability is when the non-determinism is unambiguous - when the algorithm accepts in at most one of the non-deterministic paths. The class of problems which can be solved by such restricted non-deterministic algorithms using only log-space is called Unambiguous Log-space (UL). Under a non-uniform polynomial-sized advice, the reachability problem is known to have a UL algorithm[11], thus showing $\mathsf{NL/poly} = \mathsf{UL/poly}$. Central to arriving at this complexity theoretic result was the following algorithmic result that Reinhardt and Allender [11] had established: testing reachability in a graph G augmented with a log-space computable weighting scheme that maps $w : E \to \mathbb{N}$ such that there is a unique minimum-weight path from s to any vertex v in the graph, can be done by a non-deterministic log-space algorithm unambiguously and hence is in the complexity class UL. (We call such weighting schemes as *min-unique* weighting schemes.) This also led to other important developments including an unambiguous log-space algorithm for directed planar reachability [4] - which was achieved by designing a log-space computable min-unique weighting scheme for reachability in grid-graphs (a special class of planar graphs for which reachability is as hard as planar DAG reachability[2]). An important open problem in this direction is to design a log-space min-unique weighting scheme for general graphs. The UL-computable version of this is also known to be equivalent to showing $\mathsf{NL} = \mathsf{UL}$.

Our Results: We make further progress on this algorithmic front by relaxing the restriction on the number of paths of minimum weight from one to polynomially many paths. We call a weighting scheme a *min-poly weighting scheme* if it results in at most polynomially many (in terms of $n = |V|$) paths of minimum weight from s to any vertex v in a graph $G(V, E)$.

▶ **Theorem 1.** *Testing reachability in layered DAGs, augmented with log-space computable min-poly weighting schemes, can be done by a non-deterministic log-space algorithm unambiguously and hence is in the complexity class* UL.

Our algorithms use a technique of triple inductive counting. The inductive counting method was originally discovered and employed as an algorithmic technique in [7] and [12] in order to design non-deterministic log-space algorithms for testing non-reachability in graphs. A double inductive version of this was used again by Reinhardt and Allender [11] for designing unambiguous non-deterministic algorithms for testing reachability in *min-unique* graphs. We use a triple inductive version of the inductive counting method, keeping track of one extra parameter (which is the sum of the number of minimum weight paths to each vertex). Along with a hashing technique (also used in [6]), this leads to a non-deterministic algorithm where each accepting configuration has at most one path leading to it on any input (the corresponding complexity class is known as FewUL). Finally, we convert this algorithm to a UL algorithm using an unambiguous complementary verification, thus completing the proof of the theorem.

A natural complementary question is if similar complexity bounds hold in the case of graphs with weighting assignments that result in unique maximum weight paths from s to any vertex v (such weighting schemes are called *max-unique* weighting schemes). In [8], the longest path problem on DAGs augmented with max-unique weighting assignments and having a unique sink t, was shown to be in UL. The corresponding weighting scheme with polynomially many paths of maximum weight will be called a *max-poly* weighting scheme. Using our triple inductive and complementary verification techniques, we adapt their algorithms to improve their results by relaxing the constraint on the weighting assignments - from max-unique to max-poly. We present our theorem in terms of the reachability problem, as we also show a reduction (Lemma 5) from the reachability problem to the longest path problem on single source DAGs, where the max-poly property of the graph is preserved.

▶ **Theorem 2.** *Testing reachability in layered DAGs augmented with log-space computable max-unique weighting schemes, can be done by a non-deterministic log-space algorithm unambiguously and hence is in the complexity class* UL*.*

▶ Remark. Observing that Theorem 1 and Theorem 2 hold even when the min-poly weighting scheme is UL-computable, and combining with the results of [9], it follows that: for any graph G there is a UL-computable min-poly weighting scheme if and only if there is a UL-computable min-unique weighting scheme. We also remark that, by a minor variant the proof technique in [9], we can show (the details are deferred to the appendix) that showing NL = UL is equivalent to designing UL-computable (min)max-unique weighting schemes which, thus, is equivalent to designing UL-computable (min)max-poly weighting schemes. However, we stress the importance of this relaxation of the constraints from uniqueness as this potentially can help designing weighting schemes for arbitrary layered DAGs.

Related Work: An important comparison of our results is with a complexity theoretic collapse result shown by [6]. FewL is the class of problems that has non-deterministic algorithms with only polynomially (in n) many accepting paths on any input of length n. Clearly, FewL contains all problems in UL - however, the converse is not known. In its algorithmic flavor, this question asks if reachability in a graph with at most polynomially many paths from s to t, can be done by a non-deterministic algorithm in log-space, producing at most one accepting path. ReachUL and ReachFewL are the corresponding complexity classes where the uniqueness and polynomially boundedness constraints are respectively applied for the number of paths from s to any other $v \in V$. Clearly, ReachUL is contained in ReachFewL and they were shown to be equal recently [6]. It is worthwhile noting that this establishes unambiguous log-space algorithms for reachability in graphs where there are only polynomially many paths from the start vertex to any vertex in the graph. The class of graphs that we discussed above (min/max-poly) also includes such graphs trivially. By assigning a weight of 1 to every edge in such a graph, there can only be polynomially many paths of minimum(or maximum) weight. Theorem 2, in particular, implies UL algorithms for reachability in graphs with max-unique weighting schemes where there need not exist a unique sink in the graph (and hence is a strengthening of the results in [8]).

2 Preliminaries

We assume basic familiarity with standard space complexity classes and reductions (see [3] for a standard textbook). The graphs considered in this paper are directed, acyclic and layered. Building on the terminology from the introduction, we say a DAG, $G(V, E)$, is min(max)-unique if it is augmented with a min(max)-unique weighting scheme. Similarly, a graph is said to be min(max)-poly if it is augmented with a min(max)-poly weighting scheme. A graph augmented with a weighting scheme $w : E \to \mathbb{N}$, can be converted to an un-weighted graph, by replacing each edge $e \in E$ with a path of length $w(e)$. Notice that this new graph also can be layered in log-space with edges allowed to jump forward, skipping layers arbitrarily. In particular, there is a log-space computable numbering for the vertices such that for each $(u, v) \in E$, u is given a smaller number as label than v. Additionally, in the algorithms presented in later sections, we also verify whether the input graph is min-poly and max-poly respectively.

In this new graph, we encode paths using numbers in the following way. Consider a path of length $k - 1$, $p : (x_1, x_2, \ldots, x_k)$ where the x_is are the distinct integers representing the vertices in the path. Let us represent this path p with the integer $w_p := 2^{x_1} + 2^{x_2} + \ldots + 2^{x_k}$.

In other words, each path is represented by an n-bit integer, where the ith bit is 1 if and only if vertex i is in the path. Observe that, since the graph is a layered DAG, edges are always directed from a vertex of lower index to a vertex of higher index. Thus, a set of vertices is enough to represent a path, irrespective of their order. Hence, each path p can be represented by the unique number w_p. In the case of min(max)-poly graphs, the algorithm cannot store all $s \rightsquigarrow v$ paths to check whether they are different from each other or not. Hence, we use the following hashing technique. For $v \in V$, let P_v be a set of min(max)-length $s \rightsquigarrow v$ paths. P_s, by convention, contains one $s \rightsquigarrow s$ path of length 0. Let $S_v = \{w_p \mid p \in P_v\}$. Clearly, $|S_v| = |P_v| \leq n^c$.

Hashing the weights of paths: For any path $p : s \rightsquigarrow v$, we define $\phi_m(p) := (\sum_{u \in p} 2^u) \mod m$. We say that any integer m is *good* for a vertex $v \in V$, if no two $s \rightsquigarrow v$ paths p_1 and p_2 satisfy $\phi_m(p_1) = \phi_m(p_2)$. We say that m is *good* for a graph G, if it is *good* for all $v \in V$. The following proposition ensures that there is always a polynomial sized good m.

▶ **Proposition 1.** *[5] For every constant c there is a constant c' so that for every set S of n-bit integers with $|S| \leq n^c$ there is a $c' \log n$-bit prime number m so that for all $x, y \in S$, $x \neq y \implies x \not\equiv y \mod m$.*

Guessing paths in lexicographic order: Our algorithms often require guessing several paths to a vertex v in sequence and checking whether the guessed paths are in lexicographic order w.r.t ϕ_m. Here, we outline a method of doing this in log-space.

Keep a counter c of $\log \ell$ bits to keep track of how far we have traversed along a path. Initialize this to 0. Keep $\log n$ bits to store the current vertex ρ of the current path π. Let π' be the previous path. Keep two variables, δ_π and $\delta_{\pi'}$, of $\log m$ bits each. to store the intermediate value of $\phi_m(\pi)$ and previously calculated final value of $\phi_m(\pi')$ respectively. Repeat the following two steps until c reaches ℓ. (1) $\delta_\pi = (\delta_\pi + 2^\rho) \mod m$. (2) Increment c and choose one of ρ's neighbour vertices non-deterministically and replace ρ by this neighbour.

Setting δ_π to $\delta_{\pi'}$ and setting δ_π, ρ and c to 0, repeat the steps in the previous paragraph till we have guessed all the q paths. Each time, before updating $\delta_{\pi'}$, check if δ_π is strictly less than $\delta_{\pi'}$. If not, reject there itself.

Now we fix some notation. For any vertex $v \in V$, we denote by $d(v)$ (and $D(v)$), the minimum-distance (and maximum distance) of v from s. For any vertex $v \in V$, $p(v)$ (and $P(v)$) is the number of minimum-length (and maximum-length) $s \rightsquigarrow v$ paths.

3 FewUL Algorithm for Reach in min-poly layered DAGs

The UL algorithm given by Reinhardt and Allender [11] solves Reach for min-unique graphs. In this section, we introduce a modification of their algorithm to work for min-poly graphs. To handle polynomially many minimum-length paths, we introduce a new inductive parameter p_k which stores the sum of the number of minimum length paths from s to every vertex v with $d(v) \leq k$. To inductively compute this new parameter for each k, we will use the method of guessing paths p in lexicographic order with respect to their hashed values (ϕ_m) assuming that the guess of m is *good*.

However, we are still faced with the problem of obtaining a *good* m. In the following set of routines, we will guess the value of m and use it while simultaneously detecting if it is not *good*. Note that this routine will not be unambiguous any more, because there could be several choices of m which are *good* for the given graph. However, each choice of m will lead

Algorithm MAIN-MIN-FEWUL: Main FewUL routine to check reachability on min-poly graphs.

```
1: Input: $(G, s, t)$
2: Non-deterministically guess $2 \le m < n^{c'}$
3: $k := 1$
4: $c_0 := 1; \Sigma_0 := 0; p_0 := 1$
5: $(c_1, \Sigma_1, p_1) = $ UPDATE-MIN$(G, s, 0, c_0, \Sigma_0, p_0, m)$
6: while $k < n - 1$ and $(c_{k-1}, \Sigma_{k-1}, p_{k-1}) \neq (c_k, \Sigma_k, p_k)$ do
7:    $(c_{k+1}, \Sigma_{k+1}, p_{k+1}) = $ UPDATE-MIN$(G, s, k, c_k, \Sigma_k, p_k, m)$
8:    $k := k + 1$
9: end while
10: if TEST-MIN$(G, s, t, k, c_k, \Sigma_k, p_k, m) > 0$ then
11:    Go to state ACCEPT-m
12: else
13:    REJECT
14: end if
```

Algorithm UPDATE-MIN: Deterministic (barring TEST-MIN calls) routine computing c_{k+1}, Σ_{k+1} and p_{k+1}.

```
1: Input: $(G, s, k, c_k, \Sigma_k, p_k, m)$
2: Output: $c_{k+1}, \Sigma_{k+1}, p_{k+1}$
3: $c_{k+1} := c_k$; $\Sigma_{k+1} := \Sigma_k$; $p_{k+1} := p_k$;
4: $num := 0$;
5: for $v \in V$ do
6:    if TEST-MIN$(G, s, v, k, c_k, \Sigma_k, p_k, m) = 0$ then
7:       for $x$ such that $(x, v) \in E$ do
8:          $num := num +$ TEST-MIN$(G, s, x, k, c_k, \Sigma_k, p_k, m)$;
9:          if $num > n^c$ then
10:            REJECT
11:         end if
12:      end for
13:      if $num > 0$ then
14:         $c_{k+1} := c_{k+1} + 1; \Sigma_{k+1} := \Sigma_{k+1} + k + 1; p_{k+1} := p_{k+1} + num$;
15:      end if
16:   end if
17: end for
```

to exactly one accept state. Hence, we can label these accept states with their respective choices of m, thus making it a FewUL routine.

Algorithm: Here we give the outline of the FewUL algorithm for $L = \{ (G(V,E), s, t) \mid \exists s \rightsquigarrow t$ path and $\forall v \in V, p(v) \le n^c\}$, where the value of c is known. We fix some basic notations. $c_k = |\{v \in V : d(v) \le k\}|$, $\Sigma_k = \sum_{d(v) \le k} d(v)$. The extra parameter p_k is equal to $\sum_{d(v) \le k} p(v)$. First, building on the central idea of [11], we design an unambiguous log-space routine (TEST-MIN) to determine if $d(v) \le k$ and return $p(v)$ (in at most one non-deterministic path), assuming the correct values of c_k, Σ_k, p_k and m. The modification is that, for each vertex $x \in V$ the algorithm will guess the number of paths (q - in the algorithm $q = 0$ is interpreted as "guessing that $d(v) > k$") from s to x, their length ℓ, and the paths themselves in strictly decreasing order with respect to ϕ_m. Using this subroutine, we then compute inductively, the values of c_{k+1}, Σ_{k+1} and p_{k+1}. We will inductively compute $p(v)$ and check if it is greater than the polynomial bound n^c. If $p(v)$ exceeds this number, the subroutine rejects as the underlying graph is not min-poly. This is described in the pseudocode UPDATE-MIN. The main FewUL algorithm will inductively compute c_k, Σ_k and p_k starting from $k = 1$ to $n - 1$.

Algorithm TEST-MIN: Unambiguous Log-space routine to return $p(v)$ if $d(v) \leq k$ (returns 0 if $d(v) > k$, rejects if $p(v) \geq n^c$), given correct values of c_k, Σ_k, p_k and a *good* m.

```
Input: $(G, s, v, k, c_k, \Sigma_k, p_k, m)$
$count := 0; sum := 0; paths := 0; paths.to.v := 0;$
for $x \in V$ do
  Nondeterministically guess $0 \leq q \leq n^c$
  if $q \neq 0$ then
    Nondeterministically guess $0 \leq \ell \leq k$
    Nondeterministically guess $q$ paths $p_1, p_2, \ldots p_q$ of length exactly $\ell$ each from $s$ to $x$.
    if ( $(\exists i < j, \phi_m(p_i) \leq \phi_m(p_j))$ OR (paths are not valid) ) then
      REJECT
    end if
    $count := count + 1; sum := sum + \ell; paths := paths + q;$
    if $x = v$ then
      $paths.to.v := q;$
    end if
  end if
end for
if $count = c_k$, $sum = \Sigma_k$ and $paths = p_k$ then
  Return the value of $paths.to.v$
else
  REJECT
end if
```

► **Claim 1.** *If m is* good*, given the correct values of c_k, Σ_k and p_k, the algorithm* TEST-MIN *has exactly one non-rejecting path, and it returns the correct value of $p(v)$.*

Proof. We argue that, since m is *good*, there is a unique way to guess the $d(v)$ and $p(v)$ ($\forall v \in V$), to satisfy $count = c_k$, $sum = \Sigma_k$ and $paths = p_k$. We analyze this by cases.

If the algorithm, in a non-deterministic choice, guesses $q > 0$ for some vertex v (i.e. $d(v) \leq k$) for which $d(v) > k$, then it will not be able to guess any path of length $\leq k$, and hence will end up rejecting in that non-deterministic choice. If it guesses $q = 0$ for some vertex v (i.e. $d(v) > k$) for which $d(v) \leq k$, it will not increment *count*. But then, to compensate this loss, for *count* to reach c_k, the algorithm, in this non-deterministic choice, will have to guess $q > 0$ for some vertex u for which $d(u) > k$, and thereby will reject.

If the algorithm, in a non-deterministic choice, guesses $\ell < d(v)$ ($q > p(v)$) for any v, then it will not be able to find - a path of such length (that many paths) and hence will end up rejecting in that non-deterministic choice. If it guesses $\ell > d(v)$ ($q < p(v)$), then to compensate, it will have to guess $\ell < d(u)$ ($q > p(u)$) for some other vertex u, and hence will reject in that non-deterministic path.

Hence, only the path in which, for all vertices, q and ℓ are guessed correctly and all q paths of length ℓ are guessed in lexicographical order w.r.t. ϕ_m, will be a non-reject path and will return the value of $p(v)$ correctly. ◄

► **Claim 2.** *If the algorithm* TEST-MIN *works correctly for parameter k, then given the correct values of c_k, Σ_k and p_k, the algorithm* UPDATE-MIN *computes the correct values of c_{k+1}, Σ_{k+1} and p_{k+1}.*

Proof. The algorithm first assigns $c_{k+1} := c_k, \Sigma_{k+1} := \Sigma_k$ and $p_{k+1} := p_k$. Now, to update these values we need the exact set of vertices with $d(v) = k + 1$. The algorithm, for each v, checks if $d(v) > k$ and for each of its neighbours x, checks if $d(x) \leq k$. For the neighbours passing this test, we know that $d(x) = k$. If any of the neighbours passes the test ($num > 0$ in line 13), $d(v) = k + 1$. Hence, c_{k+1} is incremented by 1, Σ_{k+1} is incremented by $k + 1$, and p_{k+1} is incremented by $\sum_{(x,v)\in E, d(x)=k} p(x)$ (which is stored in num after loop 7-12). Hence all the three parameters get updated correctly and hence the proof. ◀

▶ **Observation 1.** *Observe that, since we begin with the correct values of c_0, Σ_0 and p_0, by induction, Claims 1 and 2 imply that the values of c_k, Σ_k and p_k calculated at any time in the algorithm are always correct.*

▶ **Claim 3.** *If m is* good*, the algorithm* MAIN-MIN-FEWUL *has at most one path to state ACCEPT-m.*

Proof. Using Observation 1 and Claim 1, we know that there is exactly one non-rejecting path in each call to TEST-MIN. Thus, there is exactly one non-rejecting path in each call to UPDATE-MIN, as UPDATE-MIN is deterministic barring the calls to TEST-MIN. Similarly, there is exactly one non-rejecting path in MAIN-MIN-FEWUL, as MAIN-MIN-FEWUL - for a particular choice of m - is deterministic barring the calls to UPDATE-MIN. If t is indeed reachable from s, this non-rejecting path goes to ACCEPT-m, as m is guessed initially and is not changed thereafter. ◀

▶ **Claim 4.** *If m is not* good*, given the correct values of c_k, Σ_k and p_k, the algorithm* TEST-MIN *(and hence both* UPDATE-MIN *and* MAIN-MIN-FEWUL*) always rejects.*

Proof. If m is not *good*, then there exists a vertex v such that there exist at least two $s \rightsquigarrow v$ paths p_1 and p_2 for which $\phi_m(p_1) = \phi_m(p_2)$. So, if we guess $q = p(v)$, then the paths cannot be in strictly decreasing order w.r.t. ϕ_m and the algorithm will reject. If we guess $q > p(v)$, then the algorithm will fail to find q paths and reject. If we guess $q < p(v)$, then $paths$ will never be equal to p_k, as the q for some other vertex u will then need to be greater than $p(u)$ (for $paths$ to become equal to p_k), which is not possible. ◀

▶ **Theorem 3.** *The algorithm* MAIN-MIN-FEWUL *is correct and* FewUL*.*

Proof. If the value of m guessed is not *good*, then the algorithm MAIN-MIN-FEWUL always rejects (by Claim 4 and Observation 1), and if it is *good*, there is at most one path which reaches ACCEPT-m (Claim 3). As there are polynomially many possible values of m, MAIN-MIN-FEWUL is in FewUL. After covering all the reachable vertices, the *while* loop (line 6-9) in MAIN-MIN-FEWUL terminates with correct values of c_k, Σ_k and p_k (Observation 1) and before reaching ACCEPT-m we do a final check to see whether or not vertex t has been covered. As this case occurs only when m is *good* (Claims 3 and 4), the correct values of $p(v)$ will be returned (Claim 1) and thus the final decision will be correct. ◀

4 UL Algorithm for Reach in min-poly layered DAGs

The algorithm presented in the previous section is not unambiguous because there can be more than one *good* m. To address this, we modify the MAIN-MIN-FEWUL routine in such a way that we always use the least *good* m (let us call this integer f). The TEST-MIN and UPDATE-MIN routines are already unambiguous and need no change.

The idea is to non-deterministically guess f, and to verify that f is the smallest *good* integer for the graph G. This is done by running an unambiguous routine which checks all integers $m < f$ and, for each value, verifies that it is not *good* and proceeds to the next value. Finally it reaches f, and accepts if and only if it is *good* and there is a path from s to t.

Algorithm UPDATE-FAULT-MIN: UL routine to verify our choice of f.

```
Input: $(G, s, m)$
non-deterministically guess $1 < k_1 < n$
$c_0 := 1; \Sigma_0 := 0; p_0 := 1; k := 1$
while $k < k_1$ do
  $(c_k, \Sigma_k, p_k) = $ UPDATE-MIN$(G, s, k, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m)$
  $k := k + 1$
end while
$match\_found := false$
for $v \in V$ do
  if TEST-MIN$(G, s, v, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m) = 0$ then
    $valid := false$
    for $x$ such that $(x, v) \in E$ do
      if TEST-MIN$(G, s, x, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m) > 0$ then
        $valid := true$
      end if
    end for
    if $valid$ then
      for $(a, b)|(a, v)$ and $(b, v) \in E$ do
        $\alpha :=$ TEST-MIN$(G, s, a, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m)$
        $\beta :=$ TEST-MIN$(G, s, b, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m)$
        if $(\alpha > 0) \wedge (\beta > 0) \wedge ($FIND-MATCH$(G, s, k, a, b, \alpha, \beta, m) = true)$ then
          RETURN
        end if
      end for
    end if
  end if
end for
REJECT
```

If an integer $m < f$ is not *good*, there must be a least integer $k_1(m)$ (from s) such that there exists a vertex v for which $d(v) = k_1(m)$ and for which m is not *good*. It suffices to find this vertex in order to certify that m is not good. For any such vertex v, there must exist $a, b \in V$ such that a, b are in-neighbours of v at distance $k_1(m) - 1$ from s and there must be two paths, p_a through a and p_b through b such that $\phi_m(p_a) = \phi_m(p_b)$. Indeed, $a \neq b$, since otherwise it contradicts the choice of $k_1(m)$. This is done by an unambiguous non-deterministic algorithm FIND-MATCH$((G, s, k, a, b, \alpha, \beta, m)$, which guesses α(respectively β) number of $s \rightsquigarrow a$ ($s \rightsquigarrow b$) paths and pairwise checks for collision with respect to ϕ_m between $s \rightsquigarrow a$ and $s \rightsquigarrow b$ paths. This is used as a subroutine in UPDATE-FAULT-MIN.

▶ **Theorem 4.** *The algorithm* MAIN-MIN-UL *is correct and unambiguous* log*-space.*

Proof. Let f' be the smallest *good* value for graph G. We first argue that, if m is not *good* then there exists exactly one non-reject path in UPDATE-FAULT-MIN. We do this by considering the following cases : If $k_1 > k_1(m)$, then in the while loop (lines 4-7), when $k = k_1(m)$, UPDATE-MIN will find two paths p_1 and p_2 satisfying $\phi_m(p_1) = \phi_m(p_2)$ and will reject. If $k_1 < k_1(m)$ then FIND-MATCH will never find two paths p_1 and p_2 satisfying $\phi_m(p_1) = \phi_m(p_2)$. So, it will always return $false$ and thus, UPDATE-FAULT-MIN will reject at line 28. If $k_1 = k_1(m)$: let u be the lexicographically first vertex such that there exist two $s \rightsquigarrow u$ paths p_1 and p_2 satisfying $\phi_m(p_1) = \phi_m(p_2)$. Hence, in line 22, when $v = u$, the algorithm will return, and this is the only non-reject path.

Now we argue that, if m is *good* then UPDATE-FAULT-MIN rejects. Notice that, irrespective of the value of k_1 guessed, FIND-MATCH will not be able to find two paths p_1 and p_2 such that $\phi_m(p_1) = \phi_m(p_2)$ as m is *good*. Hence, in line 22, UPDATE-FAULT-MIN algorithm will never return and thus will reject in line 28.

Now we are ready to argue unambiguity of MAIN-MIN-UL. More specifically, we argue that if $f = f'$, MAIN-MIN-UL accepts in at most one path, and if $f \neq f'$, MAIN-MIN-UL rejects. Consider the case $f = f'$. In each iteration of the first while loop (lines 4-7) in MAIN-MIN-UL, m is not *good* and thus by the above argument, the while loop terminates in exactly one path. The rest of the algorithm (lines 8-19) is identical to MAIN-MIN-FEWUL. So, by Claim 3 there is at most one accept path. Note that here, unlike in MAIN-MIN-FEWUL, we will reach a unique accept state corresponding to $m = f = f'$.

Now consider $f \neq f'$. If $f < f'$, then at line 7, when the first while loop terminates, $m = f < f'$, and UPDATE-MIN with f as parameter will reject because of Claim 4 and Observation 1. If $f > f'$, then when in first while loop $m = f'$ (and hence m is good), UPDATE-FAULT-MIN will reject (as shown above).

Now we argue correctness. As argued, line 15 in MAIN-MIN-UL will be reached only when $f = f'$. At this point, c_k, Σ_k, p_k are calculated correctly, as Observation 1 still holds. Thus, by Claim 1, TEST-MIN outputs the correct value of $p(t)$ as $m = f'$ is *good* and thus the final result is correct. ◀

5 Reach in max-poly layered DAGs

In order to arrive at the algorithm for REACH in max-poly graphs, we solve a harder problem on a more specific class of graphs. This is a variant of the LONG-PATH problem (Given (G, s, t, j) where s and t are vertices in the graph G, and j is an integer - the LONG-PATH problem asks to check if there is a path from s to t of length at least j) where the graph G has a unique source s. We first give the reduction from REACH to this special case of LONG-PATH.

▶ **Lemma 5.** *There is a function f, computable in log-space, that transforms an instance $(G(V, E), s, t)$ of* REACH *to an instance $(G'(V', E'), s', t, 2n + 1)$ of* LONG-PATH, *where $n = |V|$, such that t is reachable from s in G if and only if there exists a path of length at least $2n + 1$ from s' to t in G'. In addition, if G is max-unique (max-poly), then G' is max-unique (max-poly).*

Proof. As mentioned in the preliminaries, without loss of generality, we can assume that the vertices of the graph $G(V, E)$ are numbered such that, edges always go from a lower numbered vertex to a higher numbered vertex. Let $V = \{v_1, v_2, \ldots, v_n\}$ be this numbering. We will construct $G'(V', E')$ as follows: In addition to the edges among the vertices in V, we add a new source vertex s' and add edges from s' to all other vertices in V. We assign weights to the newly added edges (which we later remove by replacing the edges with paths of length equal to the weight of the edge). The weight of the edge $(s', s) = 2n$ and for vertices $v_i \neq s$, weight of (s', v_i) is $2i$. Note that G' has exactly one source vertex s' and hence is a valid input for our algorithm to solve LONG-PATH.

Now we argue the that if G had a unique path (polynomially many paths) of maximum length from s to any vertex v, then so will be the case with G'. This condition is easily seen for $v \notin V$. For a vertex $v_i \in V$, we claim that among all the paths not going through s, there is exactly one path of maximum length and this is the path corresponding to the edge (s', v_i) of length $2i$. If not, choose a longest path (say p) which is not corresponding to the edge (s', v_i). Let v_j $(j < i)$ be the first vertex in p from V. Clearly, p must use the path

Algorithm MAIN-MIN-UL: Main UL routine to check reachability.

```
1: **Input**: $(G, s, t)$
2: Non-deterministically guess $2 \leq f < n^{c'}$
3: $m := 2$
4: **while** $m < f$ **do**
5:    UPDATE-FAULT-MIN$(G, s, m)$
6:    $m := m + 1$
7: **end while**
8: $k := 1$
9: $c_0 := 1; \Sigma_0 := 0; p_0 := 1$
10: $(c_1, \Sigma_1, p_1) =$ UPDATE-MIN$(G, s, 0, c_0, \Sigma_0, p_0, m)$
11: **while** $k < n - 1$ and $(c_{k-1}, \Sigma_{k-1}, p_{k-1}) \neq (c_k, \Sigma_k, p_k)$ **do**
12:    $(c_{k+1}, \Sigma_{k+1}, p_{k+1}) :=$ UPDATE-MIN$(G, s, k, c_k, \Sigma_k, p_k, m)$
13:    $k := k + 1$
14: **end while**
15: **if** TEST-MIN$(G, s, t, k, c_k, \Sigma_k, p_k, m) > 0$ **then**
16:    ACCEPT
17: **else**
18:    REJECT
19: **end if**
```

Algorithm (FIND-MATCH): UL routine to find paths with matching ϕ_m values.

```
1: **Input**: $(G, s, k, a, b, \alpha, \beta, m)$
2: **for** $i = 1$ to $\alpha$ **do**
3:    Guess a path $\pi$ of length $k - 1$ from $s$ to $a$
4:    **if** $(i \geq 2) \wedge (\phi_m(\pi) \geq X)$ **then**
5:       REJECT
6:    **end if**
7:    $X := \phi_m(\pi)$
8:    **for** $j = 1$ to $\beta$ **do**
9:       Guess a path $\pi'$ of length $k - 1$ from $s$ to $b$
10:       **if** $(j \geq 2) \wedge (\phi_m(\pi') \geq Y)$ **then**
11:          REJECT
12:       **end if**
13:       $Y := \phi_m(\pi)$
14:       **if** $X = Y$ **then**
15:          Return $true$
16:       **end if**
17:    **end for**
18: **end for**
19: Return $false$
```

corresponding to the weighted edge (s', v_j). Hence, the length of the path p can at most be $2j + (i - j) = i + j < 2i$. This contradicts the choice of p.

Thus, for a vertex $v_i \in V$ that is not reachable from s, the maximum length path in G' is unique. For a vertex $v_i \in V$ that is reachable from s, the maximum length path not through s is of weight exactly $2i$, but the paths from s' to v_i through s are of length at least $2n + 1 > 2i$. Additionally, we can see that, if there were ℓ paths of maximum length from s to any vertex v_i in G, then the number of maximum length paths from s' to v_i is also ℓ.

We now argue correctness of our reduction. Suppose that t is not reachable from s in G. In this case, none of the paths from s' to t will pass through s. Hence, using the above argument, we know that the length of any path from s' to t cannot be greater than $2n$. On the other hand, if t is reachable from s in G (say by path p), then the path (s', s) concatenated with p is a path of length $\geq 2n + 1$ from s' to t. ◀

Now we turn to this special case of the LONG-PATH problem. As mentioned in the introduction, LONG-PATH for max-unique graphs with a unique source has been studied by [8]. The UL algorithm in [8] is for LONG-PATH on max-unique graphs having a single sink t. In our version of LONG-PATH, we will consider paths from s (as opposed to paths to t in [8]) and hence we will consider only graphs with a unique source s. We will extend their algorithm to max-poly graphs, by first giving a FewUL algorithm, and then converting it to a UL algorithm using a strategy similar to the min-poly REACH algorithm in Section 4 .

5.1 FewUL Algorithm for Reach in max-poly Layered DAGs

In a way similar to our adaptation of the algorithm for min-unique graphs of [11] to work with min-poly layered DAGs, we adapt the algorithm proposed in [8] for max-unique graphs (with a unique sink) to the case for max-poly graphs with a unique source. Along with the reduction we mentioned above from REACH to LONG-PATH in such graphs (preserving the max-unique or max-poly property), this gives an algorithm for reachability testing in such graphs. We build the intuition through an example setting where the idea used in the min-poly algorithm (TEST-MIN) fails. Suppose we have the correct values of c_k, Σ_k and p_k. Even then, suppose for a vertex v, we guess $D(v) < k$ whereas actually $D(v) \geq k$. The algorithm, in this non-deterministic choice can still compensate and make it to the original summation by guessing for another u that $D(u) \geq k$ where actually $D(u) < k$. This is possible because the algorithm does not verify guesses of the kind $D(u) \geq k$ (that is, $q = 0$). In [8], this problem is addressed by introducing a new parameter $M = \sum_{v \in V} D(v)$. The value of M is also non-deterministically guessed, which if guessed correctly, will facilitate verification of the guess $D(u) \geq k$.

In a similar way, corresponding to the inductively computed parameter p_k, we introduce $P = \sum_{v \in V} P(v)$. In what follows, we will outline a FewUL algorithm with this new parameter and give a proof sketch.

Overview of the Algorithm: We introduce notation required for our exposition. We reuse c_k to denote the number of vertices $v \in V$ for which $D(v) \geq k$. $\Sigma_k = \sum_{v:D(v)<k} D(v)$, $p_k = \sum_{v:D(v)<k} P(v)$. Notice that $c_0 = n$.

We first introduce TEST-MAX$(G, s, v, c_k, \Sigma_k, p_k, m)$, which given the correct values of c_k, Σ_k and p_k, tests unambiguously whether $D(v) \geq k$ and outputs $(D(v), P(v))$ if $D(v) < k$ or outputs $(0, 0)$ if $D(v) \geq k$. We then initialize *count* $= n$ and $\sum$ and *paths* to 0. For each vertex x, we guess if $D(x) \geq k$. If we guess NO, then the algorithm runs on similar lines as TEST-MIN, where we guess the maximum path length, the number of paths of that length from s to x, and the paths themselves in strictly decreasing order with respect to ϕ_m. We decrement *count*, and increment *sum* and *paths* appropriately. If we guess YES, then we perform a similar check by guessing the maximum path length, the number of paths of that length (now at least k) from s to x, and the paths themselves in strictly decreasing order with respect to ϕ_m. However this time, we increment sum' and $paths'$ (instead of *sum* and *path*) respectively. Once we run through all the vertices, we verify the guesses of the kind $D(v) < k$ by matching *count* with c_k, *sum* with Σ_k and *paths* with p_k. In addition, we verify the guesses of the kind $D(v) \geq k$ by matching $sum + sum' = M$ and $paths + paths' = P$.

The inductive computation of c_{k+1}, Σ_{k+1} and p_{k+1} from c_k, Σ_k, and p_k is done by the routine Update-max (along the lines of Update-min). For each vertex with $D(v) = k$, it decrements c_k by 1, Σ_k by k and p_k by $\sum_{(x,v)\in E, D(x)=k-1} P(v)$ to compute c_{k+1}, Σ_{k+1} and p_{k+1} respectively. In order to find vertices with $D(v) = k$, this routine, for each node v, verifies if $D(v) = k$ by invoking the routine Test-max on v and its in-neighbours.

The main reachability test algorithm, given (G', s', t') as the input, constructs, in log-space, the instance (G, s, t, j) of the special case of Long-path problem. It runs the remaining algorithm with this new graph. The algorithm guesses m, M and P, and inductively computes c_k, Σ_k and p_k until they stabilize (which happens only at $c_k = 0$, since G is a single source graph). Finally, to answer the original reachability problem, it suffices to test if $D(t) \geq j$. Since c_k, Σ_k and p_k are available, this can be decided using the Test-max algorithm.

Proof (Sketch) of Correctness and Unambiguity. Let T and S be the correct values of M and P respectively. We claim that, irrespective of the guessed values of M and P, if the input values c_k, Σ_k and p_k are correct, then all non-reject paths of Test-max return the correct values of $P(v)$ and $D(v)$ for v if $D(v) < k$. (For other vertices it returns $(0, 0)$). If, in addition, M and P were correct and m is 'good', then there is exactly one non-reject path in Test-max and hence in Main-max-FewUL.

It can be seen that if either M or P are guessed larger than the correct value, then $sum + sum' = M$ ($paths + paths' = P$) will never be true. If at least one of them is guessed lesser than their correct value, then for the integer k such that $D(v) < k$ for all vertices $v \in V$, we will obtain $sum = \Sigma_k$ ($paths = p_k$) and $sum' = 0$ ($paths' = 0$). However, due to the correctness of the value of Σ_k (p_k), $\Sigma_k = T$ ($p_k = S$), the check $sum + sum' = M$ ($paths + paths' = P$) will fail. Hence the algorithm is correct and is FewUL. ◀

5.2 UL Algorithm for Reach in max-poly Layered DAGs

The FewUL algorithm presented in Section 3 is not unambiguous because there could be several choices of m which are *good* for G. However, there is a conceptual difficulty in guessing the lexicographically first *good* m (which we call f). Unlike in the case of min-poly graphs, here, for the each vertex $v \in V$, the guesses $D(v) \geq k$ also require verification. Suppose, $m < f$ is not *good* - i.e., there are two paths p_1 and p_2 to a vertex u with $D(u) = k_1 - 1$ (let k_1 be the least such integer) such that $\phi_m(p_1) = \phi_m(p_2)$. For any vertex x with $D(x) \geq k_1 - 1$, the value of m is not guaranteed to be *good.* Hence there could be several computation paths on which the algorithm rejects and there is no unambiguous way to skip to $m + 1$.

We outline an idea to fix this issue, which leads to the design of a UL algorithm. We defer the details to the full version of this paper. As in the case of min-poly graphs, for each m, the algorithm Update-fault-max guesses the least integer k_1 such that there is a u with $D(u) = k_1 - 1$, and two $s \rightsquigarrow u$ paths p_1 and p_2 with $\phi_m(p_1) = \phi_m(p_2)$. Prior to this point, we run Update-max with ϕ_m and ϕ_f both being calculated for the paths - and ϕ_m being computed only for the paths to vertices with $D(v) < k$. We verify whether there are two such paths with the same end point v with $D(v) = k_1 - 1$ using Find-match. In this modified algorithm, the guesses $D(v) > k$ can be verified by using ϕ_f values, since we are assuming that f is *good* (which is later verified).

If Find-match does not return *true*, the algorithm rejects. If Find-match returns *true*, then the algorithm continues in a unique path to complete the computation beyond this point, but only for f and not for m. This way, M, T and f are verified (although it is done $f - 1$ times). In the same way, we move through every $m < f$ and if the algorithm does not reject anywhere, it means our initial choice of f was correct.

References

1 Eric Allender. Reachability problems: An update. In *Proc. of CiE 2007*, pages 25–27, 2007.

2 Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theor. Comp. Sys.*, 45(4):675–723, July 2009.

3 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

4 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):4:1–4:17, February 2009.

5 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, June 1984.

6 Brady Garvin, Derrick Stolee, Raghunath Tewari, and N.V. Vinodchandran. ReachFewL = ReachUL. *computational complexity*, 23(1):85–98, 2014.

7 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, October 1988.

8 Nutan Limaye, Meena Mahajan, and Prajakta Nimbhorkar. Longest paths in planar dags in unambiguous logspace. In *Proc. of CATS 2009*, pages 101–108, 2009.

9 Aduri Pavan, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in log-space. *Computational Complexity*, 21(4):643–670, 2012.

10 Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of STOC 2005*, pages 376–385, 2005.

11 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

12 R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, November 1988.

On Bounded Reachability Analysis of Shared Memory Systems*

Mohamed Faouzi Atig[1], Ahmed Bouajjani[2], K. Narayan Kumar[3], and Prakash Saivasan[3]

1 Uppsala University, Sweden
mohamed_faouzi.atig@it.uu.se
2 LIAFA, Université Paris Diderot, France
abou@liafa.univ-paris-diderot.fr
3 Chennai Mathematical Institute, India
{kumar,saivasan}@cmi.ac.in

Abstract

This paper addresses the reachability problem for pushdown systems communicating via shared memory. It is already known that this problem is undecidable. It turns out that undecidability holds even if the shared memory consists of a single boolean variable. We propose a restriction on the behaviours of such systems, called stage bound, towards decidability. A k stage bounded run can be split into a k *stages*, such that in each stage there is at most one process writing to the shared memory while any number of processes may read from it. We consider several versions of stage-bounded systems and establish decidability and complexity results.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Reachability problem, Pushdown systems, Counter systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.611

1 Introduction

Shared memory concurrent programs are present at different levels of the software stack, from high level applications to low level software implementing system services on multicores. These programs are notoriously complex and hard to get right, which makes extremely important developing verification methods for checking their correctness. However, the design of automatic verification for these programs remains a highly challenging problem. First, it is well known that when threads can perform recursive procedure calls, the state reachability problem (which is relevant for checking safety properties) for these programs is undecidable, even when the manipulated data are finite. In the case where recursion is not allowed (or bounded), the problem is PSPACE-complete and the complexity grows exponentially in terms of the number of threads. Therefore, important issues are investigating the decidability of the state reachability problem under various assumptions on the behaviors of these programs, exploring how far the limits of decidability can be pushed, and understanding the trade-offs between behavior coverage, decidability, and complexity. This paper is a contribution addressing these issues.

To carry out our study, we adopt a formal model that is a network of processes with a shared store ranging over a finite domain, and we consider that processes can be pushdown

* This work was partially supported by the CNRS LIA Informel, the Infosys Foundation, TCS PhD Fellowship and the Uppsala Programming for Multicore Architectures Research Center (UPMARC).

systems, or 1-counter systems (seen as pushdown systems with a single element stack alphabet), or simply finite-state systems. Each of these processes may perform reads and writes on the shared store.

First, we prove that in order to get decidability of the state reachability, restricting only the data domain of the shared store is not sufficient. Indeed, we show that two parallel 1-counter systems sharing only one bit are able to encode any 2-counter machine. This result implies that, to get decidability, it is necessary to restrict the way information flows through the shared memory.

Then, the idea we consider is the following: For each computation, consider a decomposition into what we call *stages*, where in each stage only one process is unrestricted while all the others are only allowed to read. Then, we only consider computations up to some fixed bound on the number of stages. Notice that this notion of bounding, called *stage-bounding*, does not restrict the way stacks and counters are accessed. It is rather imposing that writes by different processes to the memory cannot interleave in an unbounded manner (while reads are allowed to interleave unboundedly with any kind of operations from any process).

The notion of stage-bounding is somehow inspired by the notion of context-bounding introduced by Qadeer and Rehof in [13]. However, it is clear that stage-bounding is strictly more general than context-bounding in term of behavior coverage. This is due to the fact that operations (reads and writes) by different processes can alternate unboundedly within one single stage.

Interestingly, for networks of finite-state systems, the stage-bounded analysis is NP-complete (while the unbounded analysis is PSPACE-complete as mentioned earlier). So, stage-bounded analysis in this case has the same complexity as context-bounded analysis, while it allows for significantly more coverage. However, considering networks with two pushdown systems makes stage-bounded analysis much harder. We show that for systems with precisely two pushdown systems the complexity of stage-bounded analysis is (at least) non-primitive recursive. The decidability in this case is actually still an open problem, but we can prove that for two pushdown systems and one 1-counter system the state reachability problem under stage-bounding is undecidable.

On the other hand, we prove, and this is our main result, that for networks with at most one pushdown system and any number of 1-counter systems, stage-bounded analysis is decidable, and we show that it is in NEXPTIME while it is PSPACE-hard. We establish this decidability result by a non-trivial reduction to the state reachability problem for pushdown systems with reversal-bounded counters (i.e., counters where the number of ascending and descending phases is bounded) [11], which is quite surprising since the use of the counters is unrestricted in the original system. Detailed proofs are omitted here for want of space and may be found in the full version of this paper.

Related work: Several bounding concepts have been considered in the literature in the last few years such as context-bounding and phase-bounding [12]. Stage-bounded analysis strictly generalizes context-bounded analysis, while it is incomparable with phase-bounding which is based on restricting accesses to stacks (i.e., push and pop operations by different processes in each phase) rather than restricting accesses to the shared memory. Another work based on restricting the access to stacks is for instance [1]. Again, the results there are incomparable with those we present here.

In [2], acyclic networks of communicating pushdown systems are considered. While such an acyclic network can encode computations within one stage (since in a stage information flows unidirectionally from the writer to all other processes), it has been shown that switching once between acyclic communication topologies in a network is enough to get undecidability [3]. In contrast, our main result show a case where information flow can be redirected any finite number of times.

In [8], a context-bounded analysis is proposed for a model of multithreaded programs with counters based on multi-pushdown systems with reversal bounded counters. The results of that paper are incomparable with ours since they concern different models and different analyses, and they are established using different techniques, though both works show reductions to reachability in pushdown systems with reversal bounded counters.

In [7, 6], networks of pushdown systems with non-atomic writes are considered. Atomic read-writes cannot be implemented in that model, which means that only a weak form of synchronization is possible. It is shown that for a fixed number of processes the reachability problem is undecidable, while in the parametrized case the problem becomes decidable [7] and is PSPACE-complete [6]. In contrast, our results hold even for the case where atomic read-writes are allowed and show a decidable case for a fixed number of processes. The parametrized case in the context of our stage-bounded analysis is still open and cannot be reduced to the problem considered in [7, 6].

2 Preliminaries

Let Σ be a finite alphabet. We use Σ^* and Σ^+ to denote the set of all finite words and non-empty finite words respectively over Σ; and use ϵ to denote the empty word. We also write Σ_ϵ for $\Sigma \cup \{\epsilon\}$. We let $|w|$ denote the length of the word w. A language is a (possibly infinite) set of words. Consider a word $w = a_1 \cdots a_n$ over Σ. We define the reverse word of w as $w^R := a_n \cdots a_1$. We write $w(i)$ for a_i and $w[1..j]$ for $w(1) \cdots w(j)$. We use $w_1 \cdot w_2$ or simply $w_1 w_2$ to denote the concatenation of two given words w_1 and w_2.

We define $\preceq \Sigma^* \times \Sigma^*$ to be the *sub-word relation*: For every $u = a_1 \cdots a_n \in \Sigma^*$ and $v = b_1 \cdots b_m \in \Sigma^*$, $u \preceq v$ if and only if there are $i_1, \ldots, i_n \in \{1, \ldots, m\}$ such that $i_1 < i_2 < \cdots < i_n$ and for every $j : 1 \leq j \leq n$, $a_j = b_{i_j}$. For $w \in \Sigma^*$, $\Gamma \subseteq \Sigma$, we define $w|_\Gamma \in \Gamma^*$ for the projection of w on the Γ. Given a language $L \subseteq \Sigma^*$, the *upward closure* (resp. *downward closure*) of L (w.r.t. $\preceq$) is the set $L\uparrow$ (resp. $L\downarrow$) containing all the words $w \in \Sigma^*$ such that there is a word $u \in L$ and $u \preceq w$ (resp. $w \preceq u$). Given a word $w = a_1 a_2 \cdots a_n$, we define stuttering $St(w) = a_1^+ a_2^+ \cdots a_n^+$.

3 Shared-memory Concurrent Pushdown Systems

In this section we describe the SCPS model which consists of a set of pushdown systems that communicate through shared memory.

3.1 Pushdown Systems and Counter Systems

A *pushdown system (PDS)* is a tuple $(Q, \Gamma, \Sigma, \delta, s)$ where Q is the set of states, Γ is the stack alphabet, Σ is the tape alphabet, $s \in Q$ is the initial state and δ is the transition relation. We assume that Γ contains the special bottom of stack element $\perp$. The transition set δ is a subset of $Q \times \Gamma_\epsilon \times \Sigma \times \Gamma_\epsilon \times Q$ with the restriction that if $\tau = (q, \alpha, m, \beta, q') \in \delta$ then either $\alpha = \beta = \perp$ (*emptiness test*) or $\alpha, \beta \in \Gamma_\epsilon \setminus \{\perp\}$ and $|\alpha\beta| \leq 1$. When $\beta \neq \epsilon$ and $\alpha = \epsilon$ ($\alpha \neq \epsilon$ and $\beta = \epsilon$) we say τ is a *push* (resp. *pop*) transition.

The configuration of a PDS $A = (Q, \Gamma, \Sigma, \delta, s)$ is a pair (q, γ) with $q \in Q$ and $\gamma \in (\Gamma \setminus \{\perp\})^* \perp$. The *initial configuration* is the pair $(s, \perp)$. The transition relation $\xrightarrow{a}_A$, $a \in \Sigma$, on the set of configurations is defined as follows:

1. $(q, \alpha\gamma) \xrightarrow{a}_A (q', \gamma)$ if $(q, \alpha, a, \epsilon, q') \in \delta$. Pop move.
2. $(q, \gamma) \xrightarrow{a}_A (q', \beta\gamma)$ if $(q, \epsilon, a, \beta, q') \in \delta$. Push move.
3. $(q, \gamma) \xrightarrow{a}_A (q', \gamma)$ if $(q, \epsilon, a, \epsilon, q') \in \delta$. Internal move.
4. $(q, \perp) \xrightarrow{a}_A (q', \perp)$ if $(q, \perp, a, \perp, q') \in \delta$. Emptiness test.

We omit the A and write $\xrightarrow{a}$ when A is clear from the context. We write $(q, \gamma) \xrightarrow{w} (q', \gamma')$ for $w = a_1 \dots a_n \in \Sigma^*$ to mean that there is a sequence of transitions of the form $(q, \gamma) = (q_0, \gamma_0) \xrightarrow{a_1} (q_1, \gamma_1) \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} (q_{n-1}, \gamma_{n-1}) \xrightarrow{a_n} (q_n, \gamma_n) = (q', \gamma')$. Given a configuration c, we use $L(A, c)$ to denote the set of words w such that $(s, \perp) \xrightarrow{w} c$. Given two configurations c_1, c_2, we use $L(A, c_1, c_2)$ to denote the set of words w such that $c_1 \xrightarrow{w} c_2$.

A *counter system* (CS) is a pushdown system where $\Gamma = \{\alpha, \perp\}$. In this case we refer to the push and pop moves as *increment* and *decrement* and the emptiness test as *zero test*. Finally, if the stack alphabet $\Gamma = \{\perp\}$ the PDS is just a finite state system (FSS).

3.2 Concurrent Pushdown System with Shared Memory

We consider a set of pushdown systems communicating with each other via a shared memory. The contents of this memory is drawn from a finite set $\boldsymbol{M}$ and in each move one of the pushdown systems from the collection either writes a value from $\boldsymbol{M}$ into the shared memory or reads the current value in the shared memory.

Let $\mathcal{O}_{\boldsymbol{M}} = \{\boldsymbol{!m}, \boldsymbol{?m} \mid \boldsymbol{m} \in \boldsymbol{M}\}$ denote the tape alphabet, where $\boldsymbol{!m}$ denotes writing the value $\boldsymbol{m}$ to the shared memory while $\boldsymbol{?m}$ refers to reading the value $\boldsymbol{m}$ from the shared memory. The value $\boldsymbol{m_0} \in \boldsymbol{M}$ is the initial memory value. We shall write $\mathcal{R}_{\boldsymbol{M}}$ ($\mathcal{W}_{\boldsymbol{M}}$) for the set $\{\boldsymbol{?m} \mid \boldsymbol{m} \in \boldsymbol{M}\}$ ($\{\boldsymbol{!m} \mid \boldsymbol{m} \in \boldsymbol{M}\}$). A Shared-memory Concurrent Pushdown System (SCPS) over a set of memory values $\boldsymbol{M}$ is a tuple $(\mathcal{I}, \mathcal{P}, \boldsymbol{m_0})$ where $\mathcal{I}$ is a finite set of indices and $\mathcal{P} = \{P_i \mid i \in \mathcal{I}\}$ is an $\mathcal{I}$-indexed collection of pushdown systems $P_i = (Q_i, \Gamma_i, \mathcal{O}_{\boldsymbol{M}}, \delta_i, s_i)$.

A configuration of a SCPS $(\mathcal{I}, \mathcal{P}, \boldsymbol{m_0})$ over $\boldsymbol{M}$ is a triple $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ where $\boldsymbol{q}$ assigns an element of Q_i to each $i \in \mathcal{I}$, $\boldsymbol{m} \in \boldsymbol{M}$ is the contents of the shared memory and $\boldsymbol{\gamma}$ assigns an element of $((\Gamma_i \setminus \{\perp\})^* \cdot \{\perp\})$ to each $i \in \mathcal{I}$ such that $(\boldsymbol{q}(i), \boldsymbol{\gamma}(i))$ is a configuration of P_i. The initial configuration of the system is the triple $(\boldsymbol{s}, \boldsymbol{\perp}, \boldsymbol{m_0})$ where for each i, $(\boldsymbol{s}(i), \boldsymbol{\perp}(i))$ is the initial configuration of P_i.

The transition relation $\xrightarrow{op}_i$, $op \in \mathcal{O}_{\boldsymbol{M}}, i \in \mathcal{I}$, relating configurations of the SCPS is defined as follows: $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m}) \xrightarrow{op}_i (\boldsymbol{q'}, \boldsymbol{\gamma'}, \boldsymbol{m'})$ iff $(\boldsymbol{q}(i), \boldsymbol{\gamma}(i)) \xrightarrow{op} (\boldsymbol{q'}(i), \boldsymbol{\gamma'}(i))$, $(\boldsymbol{q}(j), \boldsymbol{\gamma}(j)) = (\boldsymbol{q'}(j), \boldsymbol{\gamma'}(j))$ for $j \neq i$ and further one of the following holds
1. $op = \boldsymbol{?m}$ and $\boldsymbol{m'} = \boldsymbol{m}$ (a read operation)
2. $op = \boldsymbol{!m'}$ (a write operation)

We write $\xrightarrow{op}$ for $\biguplus_{i \in \mathcal{I}} \xrightarrow{op}_i$. This naturally extends to a relation $\xrightarrow{w}$ for $w \in \mathcal{O}^*_{\boldsymbol{M}}$. We write $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m}) \longrightarrow (\boldsymbol{q'}, \boldsymbol{\gamma'}, \boldsymbol{m'})$ if there is some $w \in \mathcal{O}^*_{\boldsymbol{M}}$ such that $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m}) \xrightarrow{w} (\boldsymbol{q'}, \boldsymbol{\gamma'}, \boldsymbol{m'})$

▶ **Remark 1.** *Communication via shared memory is* unreliable. *This is because, the reader may skip some of the values (lossiness) while reading some values multiple times (stuttering). It is easy to eliminate stuttering errors, unidirectionally, using a protocol that writes a delimiter between every adjacent pair of values. Eliminating lossiness would require acknowledgements from the reader, arranged using some a protocol (for eg. see Theorem 3).*

▶ **Remark 2.** *It is easy to extend the set of operations to include $\boldsymbol{\tau}$, indicating that the memory is not accessed, and $\boldsymbol{?m_1!m_2}$ indicating an* atomic *operation that reads the value $\boldsymbol{m_1}$ from the memory and replaces it with $\boldsymbol{m_2}$. None of the undecidability or lower-bounds proved in this paper require these instructions and the stage-bounded decidability arguments extend easily if they are included. Hence, they have been omitted here.*

4 The Reachability Problem for SCPS

A natural and important problem in verification is the control state *reachability problem*, which asks whether a particular (*bad*) control state can be reached via some execution of the system. Formally, given a SCPS $(\mathcal{I}, \mathcal{P}, \boldsymbol{m_0})$ and a configuration $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ determine whether $(\boldsymbol{s}, \bot, \boldsymbol{m_0}) \longrightarrow (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$. Unfortunately, this problem is undecidable.

▶ **Theorem 3.** *The reachability problem for SCPS is undecidable even when* $|\boldsymbol{M}| = 2$, $|\mathcal{I}| = 2$ *and both the pushdown systems in* $\mathcal{P}$ *are counter systems.*

Proof. (sketch) Fix a 2-counter machine A with two counters named 1 and 2. We construct a SCPS with two components, and we refer to them as the *master* and the *slave*. The master simulates the control state of A as well as the values of the counter 1. The job of the slave is to maintain the value of the counter 2. We show that it is possible for the master to communicate, unambiguously, a value from the set $\{1, 2, 3\}$ to the slave, standing for increment, decrement and test for zero respectively and also obtain a confirmation from the slave if it is able to complete the operation successfully. First we show how the master may communicate a single value from $\{1, 2, 3\}$ and then extend it to sequences of such values.

Assume that the memory contains the value 0. To communicate the value $i \in \{1, 2, 3\}$ the master carries out the sequence of operations $(\mathbf{!1?0})^i.(\mathbf{?1!0})^i$ on the memory. The slave guesses the value j being sent and executes a sequence of the form $(\mathbf{?1!0})^j.(\mathbf{!1?0})^j$. There are three possibilities and we analyze each of them:

1. $i = j$. In this case there is exactly one successful interleaving of the two sequences and it is of the form $(\mathbf{!1}_m\mathbf{?1}_s\mathbf{!0}_s\mathbf{?0}_m)^i.(\mathbf{!1}_s\mathbf{?1}_m\mathbf{!0}_m\mathbf{?0}_s)^i$ (where, the component involved in the memory operation is marked as a subscript). Further it leaves the memory with the value 0.
2. $i < j$. In this case, the interleaved runs reaches a deadlock after a sequence of the form $(\mathbf{!1}_m\mathbf{?1}_s\mathbf{!0}_s\mathbf{?0}_m)^i$ where both components wait for the other one to write the value 1 to proceed further.
3. $i > j$. In this case, the interleaved runs reaches a deadlock after a sequence of the form $(\mathbf{!1}_m\mathbf{?1}_s\mathbf{!0}_s\mathbf{?0}_m)^i(\mathbf{!1}_m\mathbf{!1}_s + \mathbf{!1}_s\mathbf{!1}_m)$ and both components wait for the other one to write the value 0 to proceed further.

Since all unsuccessful runs deadlock, it follows that the protocol can be repeated for any sequence of values and the system will either deadlock or succeed in communicating the sequence correctly to the slave. Finally, handling the confirmation from the slave to the master is also easy. After guessing the next operation the slave attempts to carry out the operation and only on success does it enter the protocol described above. The details are easy to formalize. ◀

5 Stage-bounded Computations

We introduce hereafter the concept of stage-bounding. We divide a run into segments, called stages, where in each stage at most one component is allowed to write on the shared memory while there is no restriction on the number of readers. We emphasize that there is no restriction placed on the number of writes or the number of context switches between the different components nor is there any restriction on the accesses to stacks during a stage. We then place an a priori bound on the number of stages in the run. Formally

▶ **Definition 4.** Let $\rho = \boldsymbol{c_0} \xrightarrow{op_1}_{p_1} \boldsymbol{c_1} \xrightarrow{op_2}_{p_2} \ldots \boldsymbol{c_{n-1}} \xrightarrow{op_n}_{p_n} \boldsymbol{c_n}$ be a run of the SCPS $(\mathcal{I}, \mathcal{P}, \boldsymbol{m})$. We say that ρ is a p-run if for all $1 \leq i \leq n$, $p_i = p$ whenever $op_i \in \mathcal{W}_{\boldsymbol{M}}$. That is, all the write transitions are contributed by the same process p.

We say that ρ is a 1-stage run if it is a p-run for some $p \in \mathcal{I}$ and a run ρ is a k-stage run if we may write $\rho = \boldsymbol{c_0} \xrightarrow{w_1} \boldsymbol{c_1} \xrightarrow{w_2} \ldots \boldsymbol{c_{k-1}} \xrightarrow{w_k} \boldsymbol{c_k}$ such that each $\boldsymbol{c_{i-1}} \xrightarrow{w_i} \boldsymbol{c_i}$ is a 1-stage run for each $1 \leq i \leq n$.

Stage-bounded Reachability Problem: Given a SCPS $(\mathcal{I}, \mathcal{P}, \boldsymbol{m_0})$, an integer k and a configuration $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ determine whether there is a k-stage run $(\boldsymbol{s}, \bot, \boldsymbol{m_0}) \longrightarrow (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$.

▶ **Remark 5.** *Stage-bounding restricts the ability to eliminate lossiness in shared-memory communication via acknowledgements (see Remark 1). This makes the undecidability of stage-bounded reachability non-trivial, in particular the proof of Theorem 7, and it is also crucial for Theorem 8.*

5.1 Stage bounded reachability for Communicating FSS

We next show that stage-bounding is relevant even when all components of the SCPS are finite-state. In this case stage bounded reachability problem is indeed easier than the unrestricted reachability problem.

▶ **Theorem 6.** *The reachability problem for an SCPS where every component is a FSS is* Pspace-complete *while the stage bounded reachability problem for SCPS where every component is a FSS is* NP-complete.

Proof. (sketch) When there is no bound on the number of stages, it is easy to see that an SCPS with n FSS components is equivalent to the product (intersection) of n FSS and hence the reachability problem is Pspace-complete.

To solve the stage bounded reachability problem, we show that it suffices to consider runs where in each stage every one of the readers participates in at most $|A_i|$ transitions, where A_i is the ith automaton. We then use this to show that in addition we may restrict to runs where in each stage the writer participates in at most $O((\sum_i |A_i|)^2)$ transitions. This immediately yields a polynomial bound on the length of stage-bounded computations to be explored to solve the reachability problem and hence a decision procedure in NP. ◀

5.2 Undecidability of Bounded-Stage Reachability

Unfortunately, stage bounding does not lead to decidability in the general case. We can indeed prove that SCPS with two pushdown systems and one 1-counter system are able to encode the computation of any Turing machine.

▶ **Theorem 7.** *The* 3*-stage reachability problem for SCPS consisting of two pushdown systems and one counter system is undecidable.*

Proof. We will reduce the halting problem for Turing machines to the stage-bounded reachability problem in a SCPS with two pushdown systems and one counter. We refer to the two pushdowns as the *generator* and the *replayer*. If somehow a writer and a reader could follow a protocol that ensures that every letter that is written is read exactly once then the undecidability would follow quite easily without the counter. However, doing this using shared memory in a stage bounded manner is tricky and details are as follows. In what follows we assume that stuttering errors are eliminated using a suitable delimiter (see Remark 1).

The simulation of a (potential) accepting run of the TM is carried out in 4 steps which use 3 stages in all. We fix a suitable encoding of the configurations as a word over some alphabet Γ and assume that this alphabet does not contain the symbol #. In the first step, the generator writes down a (initial) configuration C_1 of the TM in its stack followed by the # symbol. While doing so, it uses the shared memory to send a value, say \$, to the counter for each letter in C_1. The counter counts the number of such values. Since stuttering has been eliminated, the value of the counter c_1 is $\leq |C_1|$ at the end of this step.

In step 2, the generator guesses a sequence of configurations C_2,C_3,... C_n ending in an accepting configuration, writes them down, separated by #s, in its stack. It also writes the same sequence to the memory, as it is generated, which in turn is read by the replayer and copied on to its stack. At the end of step 2, the contents of the generator's stack is $C_n^R\#C_{n-1}^R\#\ldots C_1^R$ while that of the replayer is $y = D_m^R\#D_{n-1}^R\#\ldots D_1^R$, $m \leq n-1$ and y is a subword of $C_n^R\#C_{n-1}^R\#\ldots C_2^R$. It indicates the end of this stage by writing some suitable value to the memory which signals the end of this stage to the replayer and the counter. In all we have used one stage so far.

In step 3, the counter sends its value c_1 to the generator using the shared memory by writing c_1 copies of some fixed value ending with some special value to indicate the completion of this sequence. The generator removes one non-# symbol from his stack for each such value. At the end of this sequence of operations if the top of stack is not a # the generator will reject this run. Thus, a successful completion of this step will mean that $|C_n| \leq c_1$ and thus, $|C_n| \leq |C_1|$. At the end of this step, the contents of the generator's stack is $C_{n-1}^R\#C_{n-2}^R\#\ldots C_1^R$ and the counter is empty. This constitutes the second stage.

In the last step, the replayer removes the contents of its stack one element at a time and writes the removed value to the shared memory for the generator to read. It writes a special end marker at the end of the sequence and enters an accepting state. The sequence read by the generator would therefore be of the form $z = E_p^R\#E_{p-1}^R\#\ldots E_1^R$ (followed by the end marker) where $p \leq m \leq n-1$. Clearly z is a subword of y. The generator, as it reads E_p^R removes symbols from its stack verifying that C_{n-1} may be reached in one step from the configuration E_p (we write $E_p \Rightarrow C_{n-1}$ to indicate this), entering a reject state if either this is false or if they are not of the same length. It then repeats this procedure for E_{p-1} and C_{n-2} and so on. It enters an accepting state only if it empties its stack at the end of the entire sequence.

Observe that if the generator reaches its accepting state then p has to be $n-1$, $|E_{n-1}| = |C_{n-1}|$, ..., $|E_1| = |C_1|$ and $E_{n-1} \Rightarrow C_{n-1}$, ..., $E_1 \Rightarrow C_1$. Further, since z is a subword of y, y is a subword of $C_n^R\#C_{n-1}^R\#\ldots C_2^R$ and $p = n-1$, we have $E_i \preceq D_i \preceq C_{i+1}$ for all $1 \leq i \leq n-1$. Thus,

$$|C_1| = |E_1| \leq |C_2| = |E_2| \leq \ldots \leq |C_{n-1}| = E_{n-1} \leq |C_n|\,.$$

But $|C_n| \leq |C_1|$ and thus,

$$|C_1| = |E_1| = |C_2| = |E_2| \ldots = |C_{n-1}| = |E_{n-1}| = |C_n|\,.$$

Therefore $E_1 = C_2$, $E_2 = C_3$, ..., $E_{n-1} = C_n$ and the result follows. ◀

6 Decidability for single pushdown plus counters

We present in this section our main result:

▶ **Theorem 8.** *The stage bounded reachability problem for SCPS with at most one pushdown system is in* NEXPTIME.

Basically, we show that each counter system can be simulated by an exponential sized bounded-reversal counter system thus reducing the problem to reachability in a *pushdown automaton*[1] (PDA) with reversal bounded counters (which is known to be in NP).

The proof proceeds in three steps. The first step is applicable to any SCPS. In this step, we eliminate the shared memory, decouple the different pushdown systems as a collection of pushdown automata (PDA) and reduce the reachability problem for the SCPS to the emptiness of the intersection of these PDAs. (This problem, in general, is undecidable, but we will be able to restrict ourselves to the case where the PDAs are of a restricted variety.) In a shared memory system, the sequence of values written by the writer in a stage is not transmitted with precision to the reader as the reader may miss some values while reading others multiple times and this is what permits the decoupling.

We fix an SCPS $S = (\mathcal{I}, \mathcal{P}, \boldsymbol{m_0})$ over the set of memory values $\boldsymbol{M}$ where $\mathcal{P} = \{P_i \mid i \in \mathcal{I}\}$ is an $\mathcal{I}$ indexed collection of pushdown systems $P_i = (Q_i, \Gamma_i, \mathcal{O}_{\boldsymbol{M}}, \delta_i, s_i)$, for the rest of this section. For the moment, consider one stage runs where $p \in \mathcal{I}$ identifies the writer. Suppose we are interested in the existence of one stage runs starting at the configuration $((s_i)_{i\in\mathcal{I}}, (\rho_i)_{i\in\mathcal{I}}, \boldsymbol{m})$ and ending at some configuration $((q_i)_{i\in\mathcal{I}}, (\gamma_i)_{i\in\mathcal{I}}, \boldsymbol{m'})$. Now, consider the languages L_i, $i \in \mathcal{I}$, $i \neq p$, defined as

$$L_i = \{\boldsymbol{m_1 m_2 \ldots m_n} \mid \boldsymbol{?m_1?m_2}\ldots\boldsymbol{?m_n} \in L(P_i, (s_i, \rho_i), (q_i, \gamma_i))\}$$

and L_p given by

$$\{\boldsymbol{m.m_1 \ldots m_n.m'} \mid \boldsymbol{?m}^*\boldsymbol{!m_1?m_1}^*\boldsymbol{!m_2}\ldots\boldsymbol{!m_n?m_n}^*\boldsymbol{!m'?m'}^* \in L(P_p, (s_p, \rho_p), (q_p, \gamma_p))\}\,.$$

Then, the existence of a one stage run from $((s_i)_{i\in\mathcal{I}}, (\rho_i)_{i\in\mathcal{I}}, \boldsymbol{m})$ to $((q_i)_{i\in\mathcal{I}}, (\gamma_i)_{i\in\mathcal{I}}, \boldsymbol{m'})$ (with p as the writer) is equivalent to the non-emptiness of

$$St(L_p) \downarrow \cap \bigcap_{i \neq p} L_i \uparrow\ .$$

Moreover, the languages $St(L_p)\downarrow$ and $L_i \uparrow$ are easily realized as the languages of PDAs A_p and A_i constructed from the PDSs P_p and P_i respectively. These automata maintain the stack and control state of the PDS they simulate as well. Observe that the language accepted by these PDAs are either upward or downward closed.

We are however interested in k stage runs where the identity of the writer (and hence the closures to be applied) changes with the stage. Further, across the stage boundaries, we have to preserve the control state and stacks of each component as well as the content of the shared memory.

It is useful to work with a fixed sequence τ of length k over $\mathcal{I}$ identifying the writers in the k stages. Let $\tau := p_1, p_2, \ldots, p_k$, $p_i \in \mathcal{I}$ be such a sequence. Let $(\boldsymbol{s}, \perp, \boldsymbol{m_0})$ and $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ be the initial and target configurations of the SCPS and we wish to determine if there is a k stage run consistent with τ that goes from the initial to the target configuration. In this case, the pushdown automaton A_i^τ plays the role played by the automaton A_i in the one stage setting. It simulates P_i and its runs break up into k parts, where in the jth part it applies either a stuttering downward closure or upward closure to the behaviour of P_i depending on whether $j = \tau(j)$ or not. Notice that A_i^τ maintains the control state and stack of P_i. A_i^τ also makes explicit the boundary points between stage i and stage $i+1$ by using a letter of

[1] We plan to use "automata" instead of systems when they are used as language generators and to avoid ambiguity with the components of the SCPS.

the form $(\boldsymbol{m}, i)$ (instead of just $\boldsymbol{m}$). These marker letters allow us to *synchronize* the stage boundaries of the different A_i^{τ}'s. Further, these markers are also used to ensure that the contents are of the memory are correctly transferred across stages. We formalize these ideas below.

We use $\boldsymbol{M}_i$ (resp. $\boldsymbol{M}_{\tau}$) to denote $\boldsymbol{M}^* \cdot (\boldsymbol{M} \times \{i\})$ for all $i \in [1..k]$ (resp. $\bigcup_{i \in [1..k]} \boldsymbol{M}_i \cup \boldsymbol{M}$).

▶ **Lemma 9.** *For every $p \in \mathcal{I}$, we can construct, in polynomial time in $|S|$, a PDA A_p^{τ} over the stack alphabet Γ_p, with a target configuration c_p such that*

- *if $w \in L(A_p^{\tau}, c_p)$ then $w \in \boldsymbol{M}_1 \cdot \boldsymbol{M}_2 \cdots \boldsymbol{M}_k$. (unambiguous breakup)*
- *if $w \in L(A_p^{\tau}, c_p)$ and $w = w_1 w_2 \ldots w_k$ with $w_i \in \boldsymbol{M}_i$ for all $1 \leq i \leq k$ then $w_1'.w_2'. \ldots w_k' \in L(A_p^{\tau}, c_p)$ for all $w_1', \ldots w_k'$ such that, for all i, $w_i' \in \boldsymbol{M}_i$ and either $p = \tau(i)$ and $w_i' \in St(w_i){\downarrow}$ or $p \neq \tau(i)$ and $w_i' \in (w_i \uparrow \cap \boldsymbol{M}_i)$. (closure)*
- *There is a k stage run from $(\boldsymbol{s}, \bot, \boldsymbol{m_0})$ to $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ with $\tau(i)$ as the writer in the ith stage iff $\bigcap_{p \in \mathcal{I}} L(A_p^{\tau}, c_p) \neq \emptyset$. (decoupling)*

In the second step we exploit the fact that the language of each A_p^{τ} is a finite unambiguous concatenation of languages that are upward or downward closed. Towards this we first state two propositions which explain the importance of closures.

▶ **Proposition 10** (Downward closure of CFLs [5]). *Given a pushdown automaton P and two configurations c_i, c_f, we can construct, in time and space at most exponential in size of P, c_i and c_f, a FSA A with two configurations c_i' and c_f' such that $L(A, c_i', c_f') = L(P, c_i, c_f){\downarrow}$.*

▶ **Proposition 11** (Upward closure of CFLs). *Given a pushdown automaton P and two configurations c_i, c_f, we can construct, in time and space at most exponential in size of P, c_i and c_f, a FSA A with two configurations c_i' and c_f' such that $L(A, c_i', c_f') = L(P, c_i, c_f){\uparrow}$.*

This means that, if we are dealing with a single stage then we may replace the PDA A_i, $i \in \mathcal{I}$, described earlier, by exponential sized finite automata B_i, $i \in \mathcal{I}$ (for all i, including the writer p). Thus we have reduced the problem to the emptiness of the intersection for FAs. However the k stage case is somewhat more complex. This is because, as A_i^{τ} switches from one stage to the next, it has to preserve the configuration of P_i (i.e. the contents of the stack) as well as the contents of the memory. While this is trivial when A_i^{τ} is a pushdown, it is not possible to do this using finite number of states. However, all is not lost as we may convert A_i^{τ} into a $2k$*-turn PDA* B_i^{τ}. A run of pushdown automaton is said to be 1-*turn* if the stack height of the sequence of configurations is either uniformly non-increasing (does not involve a push move) or non-decreasing (does not involve a pop move). A k*-turn* run is concatenation of k sequences of 1-*turn* runs. A k*-turn* PDA is one which only allows at most k*-turn* runs (see [9]).

We explain the ideas behind the construction of B_i^{τ} now. Let us fix a pushdown automaton A. For any $\gamma \in \Gamma^* \bot$ we say that a run χ from (q, ρ) to (q', ρ') is a γ-run if γ is the longest suffix of ρ that appears as a suffix of the contents of the stack in every configuration along the run χ. Observe that, this implies that γ must be a suffix of ρ and ρ' and further there is a configuration in χ whose stack content is exactly γ. (Observe that every run from (q, ρ) to (q', ρ') is a γ-run for some (unique) suffix γ of ρ). We write $L_{\gamma}(c, c')$ to refer to the set of words accepted on γ-runs from c to c'. Let $c = (q, \rho)$ to $c' = (q', \rho')$. Then, $L(c, c')$, the set of words accepted on runs from c to c' is

$$\{x.y \mid x \in L_{\gamma}(c, (q'', \gamma)), y \in L_{\gamma}((q'', \gamma), c'), \gamma \text{ a suffix of } \rho, q'' \in Q\}.$$

We write $Cl(L)$ to refer to the upward or downward closure of L when the identity of the closure does not matter. Thus $Cl(L(c, c'))$ is

$$\{x.y \mid x \in Cl(L_{\gamma}(c, (q'', \gamma))), y \in Cl(L_{\gamma}((q'', \gamma), c')), \gamma \text{ a suffix of } \rho, q'' \in Q\}.$$

For each $\alpha \in \Gamma$ and $q_1, q_2 \in Q$, we let:

$$L^-_\alpha(q_1,q_2) = \{w \mid (q_1, \alpha\bot) \xrightarrow{w} (q_2, \bot) \text{ without using emptiness tests } \}$$
$$L^+_\alpha(q_1,q_2) = \{w \mid (q_1, \bot) \xrightarrow{w} (q_2, \alpha\bot) \text{ without using emptiness tests } \}$$
$$L^\bot(q_1,q_2) = \{w \mid (q_1, \bot) \xrightarrow{w} (q_2, \bot)\}$$

We can see that the language $L_\gamma(c, (q'', \gamma))$ (resp. $L_\gamma((q'', \gamma), c')$) can be rewritten as a concatenation of the following languages $L^-_{\alpha_1}(q, q_1) \cdot L^-_{\alpha_2}(q_1, q_2) \cdots L^-_{\alpha_\ell}(q_{\ell-1}, q'') \cdot L$ with $\rho = \alpha_1\alpha_2 \cdots \alpha_\ell\gamma$ (resp. $L \cdot L^+_{\alpha_1}(q'', q_1) \cdot L^+_{\alpha_2}(q_1, q_2) \cdots L^+_{\alpha_\ell}(q_{\ell-1}, q')$ with $\rho' = \alpha_\ell\alpha_{\ell-1} \cdots \alpha_1\gamma$) and $L = \{\epsilon\}$ if $\gamma \neq \bot$ and $L = L^\bot(q'', q'')$ otherwise.

Hence, any word $w \in Cl(L(c, c'))$ can be rewritten as the concatenation of three words (i.e., $w = w_1w_2w_3$). The first word w_1 is in $Cl(L^-_{\alpha_1}(q, q_1)) \cdot Cl(L^-_{\alpha_2}(q_1, q_2)) \cdots Cl(L^-_{\alpha_\ell}(q_{\ell-1}, q''))$ for some letters $\alpha_1\alpha_2 \cdots \alpha_\ell$ and stack content γ such that $\rho = \alpha_1\alpha_2 \cdots \alpha_\ell\gamma$. The second word w_2 is in $Cl(L^\bot(q'', q''))$ if $\gamma = \bot$, and in $\{\epsilon\}$ otherwise. The last word w_3 is in $Cl(L^+_{\alpha'_1}(q'', q'_1)) \cdot Cl(L^+_{\alpha'_2}(q'_1, q'_2)) \cdots Cl(L^+_{\alpha'_m}(q'_{m-1}, q'))$ for some letters $\alpha'_1\alpha'_2 \cdots \alpha'_m$ such that $\rho' = \alpha'_m\alpha'_{m-1} \cdots \alpha'_1\gamma$.

Furthermore, the languages $L^-_\alpha(q_1, q_2)$, $L^+_\alpha(q_1, q_2)$ and $L^\bot(q_1, q_2)$ are context-free and their upward and downward closures are effectively regular (see Propositions above) and so let $B^-_\alpha(q_1, q_2)$, $B^+_\alpha(q_1, q_2)$ and $B^\bot(q_1, q_2)$ be finite automata recognizing $Cl(L^-_\alpha(q_1, q_2))$, $Cl(L^+_\alpha(q_1, q_2))$ and $Cl(L^\bot(q_1, q_2))$ respectively.

Now, we describe the PDA B: It uses the same stack alphabet as A and maintains the *current* state of A as part of its local state. Its run consists of 3 phases. The first phase consists in generating the first word w_1 by repeating the following a number of times: it guesses a triple (q_1, α, q_2), verifies that q_1 is the current state of A, pops the top of stack and verifies that it is indeed α, simulates a run of $B^-_\alpha(q_1, q_2)$ and then changes the current state of A to q_2. In the optional second phase, it will generate the second word w_2 by verifying that the stack is empty, guessing a pair (q_1, q_2), checking that q_1 is the current state of A, simulating a run of $B^\bot(q_1, q_2)$ and then changing the current state of A to q_2. The third stage consists of generating the word w_3 by repeating the following a number of times: it guesses triples of the form (q_1, α, q_2), verifies that q_1 is the current state of A, pushes α on to the stack, simulates a run of $B^+_\alpha(q_1, q_2)$ and then changes current state of A to q_2. Observe that these runs involve only two turns one during the pop phase and one during the push phase. The language of B while starting at the stack configuration γ with q as the current state of P and ending with stack configuration γ' and q' as the current state of P is the language $Cl(L(c, c'))$. But, if $L(c, c')$ is already closed then $Cl(L(c, c')) = L(c, c')$. Thus, in this case B simulates a (arbitrary) run of A from c to c' using a single turn run and further maintains the configuration reached by A at the end of this run.

Since the language of A^τ_p in each stage is either upward or downward closed, by concatenating k appropriately chosen copies of the automata B (with correct closures in each stage depending on the sequence τ) we construct a $2k$-*turn* PDA B^τ_p that has the same language as A^τ_p as stated by the following Lemma

▶ **Lemma 12.** *For every $p \in \mathcal{I}$, it is possible to construct, in exponential time in the size of A^τ_p, a $2k$ turn PDA B^τ_p and a configuration c'_p, such that $L(B^\tau_p, c'_p) = L(A^\tau_p, c_p)$.*

Unfortunately, the emptiness of the intersection of even two 2-*turn* PDAs is undecidable, as can be seen from an easy reduction from the Post's correspondence problem (PCP). The situation is quite different when the PDAs are counters. In fact, we can show:

▶ **Lemma 13.** *Let k be a natural number. Let A_1 be a $2k$ turn PDA and $A_2, \ldots, A_n$ a sequence of $2k$-turn counter automata. Let c_i be a configurations of A_i for all $i : 1 \leq i \leq n$. Then, the problem of checking whether $L(A_1, c_1) \cap \cdots \cap L(A_n, c_n)$ is not empty can be decided in nondeterministic time that is polynomial in the size of A_i and k, and exponential in n.*

The proof of Lemma 13 is done by a reduction to the state reachability problem for pushdown systems with reversal-bounded counters where each counter is allowed a bounded number of alternation between *modes*. (The latter problem is known to be NP-complete [10].) A counter *mode* is a run of the system where the performed sequence of transitions on this counter consists, apart from internal transitions, only of increment transitions or only of decrement transitions or only of zero test transitions. A pushdown system with k reversal-bounded counters is pushdown system augmented with counters where any run decomposes into at most k modes for each counter. Then, the reduction consists simply in constructing a pushdown system S augmented with counters that simulates the synchronous product of $A_1, \ldots, A_n$ while observing that any run of a $2k$-*turn* counter automaton performs at most $3k$ modes in S (Note that zero tests are counted as a reversal (*mode*) but are not counted as a turn). These arguments can be formalized to obtain a complete proof.

Thus we have reduced the k stage bounded reachability problem of an SCPS consisting of one PDS and $(n-1)$ counter systems to exponentially[2] many instances of the emptiness problem for the intersection of a polynomial sized $2k$-turn PDA and $(n-1)$ $2k$-turn counter automata of exponential size (Lemma 9 and Lemma 12). We have also shown that each instance of the latter problem can be decided in Nexptime (Lemma 13). Combining these results yields to a Nexptime upper-bound and a proof of Theorem 8.

7 Lower Bounds for the Stage-bounded Reachability Problem

We have so far shown that the stage bounded reachability problem for systems with at least two pushdowns and one counter is undecidable while it can be decided in NExptime for systems containing at most one pushdown. This problem remains open for SCPS with exactly two pushdowns. In the following, we show that even if it is decidable its complexity cannot be primitive recursive.

The *regular post embedding* problem is the following: Let Σ and Γ be two alphabets. Given two functions $f : \Sigma \to \Gamma^+$ and $g : \Sigma \to \Gamma^+$, extended homomorphically to Σ^+, and a regular language $R \subseteq \Sigma^+$, does there exist a $w \in R$ such that $f(w) \preceq g(w)$? As shown in [4], this problem is decidable but cannot be solved by any algorithm with primitive recursive complexity. We reduce the regular post embedding problem to the stage-bounded reachability problem for SCPS with two pushdowns to obtain the following theorem

▶ **Theorem 14.** *The 2-stage bounded reachability problem for SCPS with two pushdowns cannot be solved by any algorithm whose complexity is primitive recursive.*

The emptiness problem for the intersection of a collection of n finite automata is known to be Pspace complete and we reduce this problem to the n-stage bounded reachability problem for SCPS with n counters to obtain the following theorem

▶ **Theorem 15.** *The stage-bounded reachability problem for SCPS consisting only of counter systems is* Pspace-Hard.

[2] The exponential blow-up comes from the guess of the sequence of writers.

8 Conclusion

We have introduced a new concept for bounding the analysis of shared memory concurrent systems. This concept is based on bounding the number of switches between writes by different processes to the shared memory, without restricting the way reads can be performed by any of the processes in the system.

Stage-bounding allows to improve significantly the behaviors coverage w.r.t. context-bounding. We have shown that for networks of finite-state systems, the complexity of stage-bounded analysis is NP-complete, as for context-bounding. In practice, this implies that this analysis can be implemented using a complete bounded model-checking with a polynomial bound. In the case of networks of infinite-state systems, we have mainly shown that the state reachability problem in networks of counter systems with a shared store is decidable under stage-bounding, and that the same still holds for networks with one additional pushdown.

Several questions remain open. One of them is closing the gap between the known upper and lower bounds on the complexity. Also, the case two pushdown systems is open, although we know that even if it is decidable, it would be with a high complexity. Finally, an interesting open question is whether it is possible to generalize our decidability result to dynamic or parametrized networks of counter systems, by considering that each component in the network is allowed to be the single writer in a bounded number of stages.

References

1 Mohamed Faouzi Atig. Model-checking of ordered multi-pushdown automata. *Logical Methods in Computer Science*, 8(3), 2012.

2 Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, volume 5201 of *Lecture Notes in Computer Science*, pages 356–371. Springer, 2008.

3 Mohamed Faouzi Atig and Tayssir Touili. Verifying parallel programs with dynamic communication structures. In *CIAA*, volume 5642 of *Lecture Notes in Computer Science*, pages 145–154. Springer, 2009.

4 Pierre Chambart and Ph. Schnoebelen. Post embedding problem is not primitive recursive, with applications to channel systems. In *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2007.

5 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.

6 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 124–140. Springer, 2013.

7 Matthew Hague. Parameterised pushdown systems with non-atomic writes. In *FSTTCS*, volume 13 of *LIPIcs*, pages 457–468. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

8 Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2012.

9 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley series in computer science. Addison-Wesley, 1979.

10 Rodney R. Howell and Louis E. Rosier. An analysis of the nonemptiness problem for classes of reversal-bounded multicounter machines. *J. Comput. Syst. Sci.*, 34(1):55–74, 1987.

11 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

12 Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.

13 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005.

Parameterized Communicating Automata: Complementation and Model Checking*

Benedikt Bollig[1], Paul Gastin[1], and Akshay Kumar[2]

1 **LSV, ENS Cachan & CNRS, France**
`{bollig,gastin}@lsv.ens-cachan.fr`
2 **Indian Institute of Technology Kanpur, India**
`kakshay@iitk.ac.in`

Abstract

We study the language-theoretical aspects of parameterized communicating automata (PCAs), in which processes communicate via rendez-vous. A given PCA can be run on any topology of bounded degree such as pipelines, rings, ranked trees, and grids. We show that, under a context bound, which restricts the local behavior of each process, PCAs are effectively complementable. Complementability is considered a key aspect of robust automata models and can, in particular, be exploited for verification. In this paper, we use it to obtain a characterization of context-bounded PCAs in terms of monadic second-order (MSO) logic. As the emptiness problem for context-bounded PCAs is decidable for the classes of pipelines, rings, and trees, their model-checking problem wrt. MSO properties also becomes decidable. While previous work on model checking parameterized systems typically uses temporal logics without next operator, our MSO logic allows one to express several natural next modalities.

1998 ACM Subject Classification F.1.1 [Computation by Abstract Devices]: Models of Computation, F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs

Keywords and phrases parameterized verification, complementation, monadic second-order logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.625

1 Introduction

The "regularity" of an automata model is intrinsically tied to characterizations in algebraic or logical formalisms, and to related properties such as closure under complementation and decidability of the emptiness problem. Most notably, the robustness of finite automata is witnessed by the Büchi-Elgot-Trakhtenbrot theorem, stating their expressive equivalence to monadic second-order (MSO) logic. In the past few years, this fundamental result has been extended to models of concurrent systems such as communicating finite-state machines (see [10] for an overview) and multi-pushdown automata (e. g., [11, 12]). Hereby, the system topology, which provides a set of processes and links between them, is usually supposed to be static and fixed in advance. However, in areas such as mobile computing or ad-hoc networks, it is more appropriate to design a program, and guarantee its correctness, independently of the underlying topology, so that the latter becomes a parameter of the system.

There has been a large body of literature on parameterized concurrent systems [9, 8, 6, 2, 1], with a focus on verification: Does the given system satisfy a specification independently of

* Supported by LIA InForMel.

the number of processes? A variety of different models have been introduced, covering a wide range of communication paradigms such as broadcasting, rendez-vous, token-passing, etc. So far, however, it is fair to say that there is no such thing as a canonical or "robust" model of parameterized concurrent systems.

This paper tries to take a step forward towards such a model. It is in line with a study of a *language theory* of parameterized concurrent systems that has been initiated in [3, 4]. We resume the model of parameterized communicating automata (PCAs), a conservative extension of classical communicating finite-state machines [5]. While the latter run a fixed set of processes, a PCA can be run on *any* topology of bounded degree, such as pipelines, rings, ranked trees, or grids. A topology is a graph, whose nodes represent processes that are connected via interfaces. Every process will run a local automaton executing send and receive actions, which allows it to communicate with an adjacent process in a rendez-vous fashion. As we are interested in language-theoretical properties, we associate, with a given PCA, the set of all possible executions. An execution includes the underlying topology, the events that each process executes, and the causal dependencies that exist between events. This language-theoretic view is different from most previous approaches to parameterized concurrent systems, which rather consider the transition system of reachable configurations. Yet, it will finally allow us to study such important concepts like complementation and MSO logic. Note that logical characterizations of PCAs have been obtained in [3]. However, those logics use negation in a restricted way, since PCAs are in general not complementable. This asks for restrictions of PCAs that give rise to a *robust* automata model. In this paper, we will therefore impose a bound on the number of *contexts* that each process traverses. We explain this notion below.

The efficiency of distributed algorithms and protocols is usually measured in terms of two parameters: the number n of processes, and the number k of contexts. Here, a context, sometimes referred to as *round*, restricts communication of a process to patterns such as "send a message to each neighbor and receive a message from each neighbor". In this paper, we consider more relaxed definitions where, in every context, a process may perform an unbounded number of actions. In an *interface*-context, a process can send and receive an arbitrary number of messages to/from a *fixed* neighbor. A second context-type definition allows for arbitrarily many sends to all neighbors, or receptions from a fixed neighbor.

In general, basic questions such as reachability are undecidable for PCAs, even when we restrict to simple classes of topologies such as pipelines. To get decidability, it is therefore natural to bound one of the aforementioned parameters, n or k. Bounding the number n of processes is known as *cut-off*. However, the trade-off between n and k is often in favor of an up to exponentially smaller k. Moreover, many distributed protocols actually restrict to a bounded number of contexts, such as P2P protocols and certain leader-election protocols. Therefore, bounding the parameter k seems to be an appropriate way to overcome the theoretical limitations of formally verifying parameterized concurrent systems.

The most basic verification question of context-bounded PCAs has been considered in [4]: Is there a topology that allows for an accepting run of the given PCA? In the present paper, we go beyond such nonemptiness/reachability issues and consider PCAs as language acceptors. We will show that, under suitable context bounds, PCAs form a robust automata model that is closed under complementation. Complementability relies on a disambiguation construction, which is the key technical contribution of the paper.

Our complementation result has wider applications and implications. In particular, we obtain a characterization of context-bounded PCAs in terms of MSO logic. Together with the results from [4], this implies that context-bounded model checking of PCAs against MSO logic is decidable for the classes of pipelines, rings, and trees. Note that MSO logic is quite powerful and, unlike in [7, 2], we are not constrained to drop any (next) modality. Actually, a variety of natural next modalities can be expressed in MSO logic, such as process successor, message successor, next event on a neighboring process, etc.

Context-bounds were originally introduced for (sequential) multi-pushdown automata as models of multi-threaded recursive programs [15]. Interestingly, determinization procedures have been used to obtain complementability and MSO characterizations for context-bounded multi-pushdown automata [11, 12]. A pattern that we share with these approaches is that of computing *summaries* in a deterministic way. Overall, however, we have to use quite different techniques, which is due to the fact that, in our model, processes evolve asynchronously.

In Section 2, we settle some basic notions such as topologies and message sequence charts, which describe the behavior of a system. PCAs and their restrictions are introduced in Section 3. Section 4 presents our main technical contribution: We show that context-bounded PCAs are complementable. This result is exploited in Section 5 to obtain a logical characterization of PCAs and decidability of the model-checking problem wrt. MSO logic. Omitted proofs can be found in the full version of the paper, which is available at: `http://hal.archives-ouvertes.fr/hal-01030765/`.

2 Preliminaries

For $n \in \mathbb{N}$, we set $[n] := \{1, \ldots, n\}$. Let $\mathbb{A}$ be an alphabet and I be an index set. Given a tuple $\bar{a} = (a_i)_{i \in I} \in \mathbb{A}^I$ and $i \in I$, we write $\bar{a}_i$ to denote a_i.

Topologies. We will model concurrent systems without any assumption on the number of processes. However, we will have in mind that processes are arranged in a certain way, for example as pipelines or rings. Once such a class and the number of processes are fixed, we obtain a topology. Formally, a topology is a graph. Its nodes represent processes, which are connected via interfaces. Let $\mathcal{N} = \{a, b, c, \ldots\}$ be a *fixed* nonempty finite set of *interface names* (or, simply, *interfaces*). When we consider pipelines or rings, then $\mathcal{N} = \{a, b\}$ where a refers to the right neighbor and b to the left neighbor of a process, respectively. For grids, we will need two more names, which refer to adjacent processes above and below. Ranked trees require an interface for each of the (boundedly many) children of a process, as well as a pointer to the father process. As $\mathcal{N}$ is fixed, topologies are structures of bounded degree.

▶ **Definition 1.** A *topology* over $\mathcal{N}$ is a pair $\mathcal{T} = (P, \longmapsto)$ where P is the nonempty finite set of *processes* and $\longmapsto \subseteq P \times \mathcal{N} \times \mathcal{N} \times P$ is the *edge relation*. We write $p \overset{a\ b}{\longmapsto} q$ for $(p, a, b, q) \in \longmapsto$, which signifies that the a-interface of p points to q, and the b-interface of q points to p. We require that, whenever $p \overset{a\ b}{\longmapsto} q$, the following hold:

(a) $p \neq q$ (there are no self loops),

(b) $q \overset{b\ a}{\longmapsto} p$ (adjacent processes are mutually connected), and

(c) for all $a', b' \in \mathcal{N}$ and $q' \in P$ such that $p \overset{a'\ b'}{\longmapsto} q'$, we have $a = a'$ iff $q = q'$ (an interface points to at most one process, and two distinct interfaces point to distinct processes).

We do not distinguish isomorphic topologies.

▶ **Example 2.** Example topologies are depicted in Figures 1 and 2. In Figure 2, five processes are arranged as a *ring*. Formally, a ring is a topology over $\mathcal{N} = \{a, b\}$ of the form $(\{1, \ldots, n\}, \longmapsto)$ where $n \geq 3$ and $\longmapsto = \{ (i, a, b, (i \bmod n) + 1) \mid i \in [n]\} \cup \{((i \bmod n) + 1, b, a, i) \mid i \in [n]\}$. A ring is uniquely given by its number of processes. Moreover, as we do not distinguish isomorphic topologies, it does not have an "initial" process. A *pipeline* is of

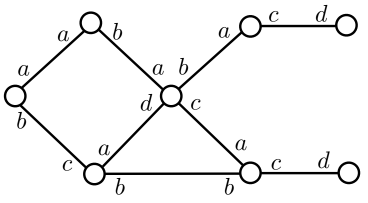


Figure 1 A topology over $\{a, b, c, d\}$.

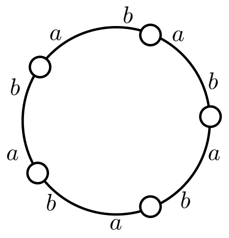


Figure 2 A ring topology.

the form $(\{1, \ldots, n\}, \longmapsto)$ where $n \geq 2$ and $\longmapsto\; = \{\, (i, a, b, i{+}1) \mid i \in [n{-}1]\} \cup \{(i{+}1, b, a, i) \mid i \in [n-1]\}$. Similarly, one can define ranked trees and grids [3]. ◀

MSO Logic over Topologies. The acceptance condition of a parameterized communicating automaton (PCA, as introduced in the next section) will be given in terms of a formula from *monadic second-order (MSO) logic*, which scans the final configuration reached by a PCA: the underlying topology together with the local final states in which the processes terminate. If S is the finite set of such local states, the formula thus defines a set of S-labeled topologies, i.e., structures $(P, \longmapsto, \lambda)$ where $(P, \longmapsto)$ is a topology and $\lambda : P \to S$. The logic $\mathrm{MSO}_t(S)$ is given by the grammar $\mathcal{F} ::= u \overset{a\ \ b}{\longmapsto} v \mid u = v \mid \lambda(u) = s \mid u \in U \mid \exists u.\mathcal{F} \mid \exists U.\mathcal{F} \mid \neg\mathcal{F} \mid \mathcal{F} \vee \mathcal{F}$ where $a, b \in \mathcal{N}$, $s \in S$, u and v are first-order variables (interpreted as processes), and U is a second-order variable (ranging over sets of processes). Note that we assume an infinite supply of variables. Given a sentence $\mathcal{F} \in \mathrm{MSO}_t(S)$ (i.e., a formula without free variables), we write $L(\mathcal{F})$ for the set of S-labeled topologies $(P, \longmapsto, \lambda)$ that satisfy $\mathcal{F}$. Hereby, satisfaction is defined in the expected manner (cf. also Section 5, presenting an extended logic).

Message Sequence Charts. Recall that our primary concern is a language-theoretic view of parameterized concurrent systems. To this aim, we associate with a system its language, i.e., the set of those behaviors that are generated by an accepting run. One single behavior is given by a message sequence chart (MSC). An MSC consists of a topology (over the given set of interfaces) and a set of events, which represent the communication actions executed by a system. Events are located on the processes and connected by process and message edges, which reflect causal dependencies (as we consider rendez-vous communication, a message edge has to be interpreted as "simultaneously").

▶ **Definition 3.** A *message sequence chart (MSC)* over $\mathcal{N}$ is a tuple $M = (P, \longmapsto, E, \lhd, \pi)$ where $(P, \longmapsto)$ is a topology over $\mathcal{N}$, E is the nonempty finite set of *events*, $\lhd \subseteq E \times E$ is the *acyclic* edge relation, which is partitioned into $\lhd_{\mathsf{proc}}$ and $\lhd_{\mathsf{msg}}$, and $\pi : E \to P$ determines the location of an event in the topology; for $p \in P$, we let $E_p := \{e \in E \mid \pi(e) = p\}$. We require that the following hold:

- $\lhd_{\mathsf{proc}}$ is a union $\bigcup_{p \in P} \lhd_p$ where each $\lhd_p \subseteq E_p \times E_p$ is the direct-successor relation of some total order on E_p,
- there is a partition $E = E_! \uplus E_?$ such that $\lhd_{\mathsf{msg}} \subseteq E_! \times E_?$ defines a bijection from $E_!$ to $E_?$,
- for all $(e, f) \in \lhd_{\mathsf{msg}}$, we have $\pi(e) \overset{a\ \ b}{\longmapsto} \pi(f)$ for some $a, b \in \mathcal{N}$, and
- in the graph $(E, \lhd \cup \lhd_{\mathsf{msg}}^{-1})$, there is no cycle that uses at least one $\lhd_{\mathsf{proc}}$-edge (this ensures rendez-vous communication).

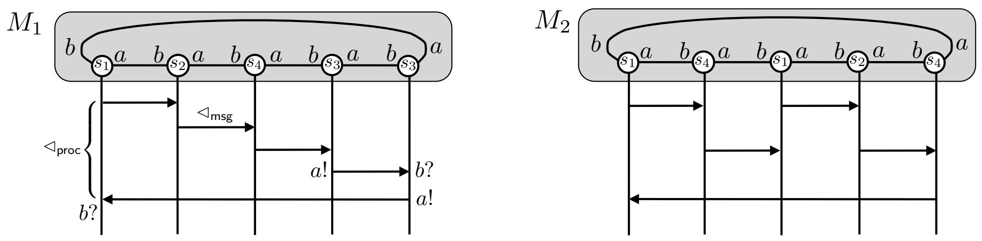


Figure 3 Two MSCs over a ring topology; local states labeling the topology in a PCA run.

Let $\Sigma = \{a! \mid a \in \mathcal{N}\} \cup \{a? \mid a \in \mathcal{N}\}$. We define a mapping $\ell_M : E \to \Sigma$ that associates with each event the type of action that it executes: For $(e, f) \in \lhd_{\mathsf{msg}}$ and $a, b \in \mathcal{N}$ such that $\pi(e) \overset{a\ \ b}{\longmapsto} \pi(f)$, we set $\ell_M(e) = a!$ and $\ell_M(f) = b?$.

The set of MSCs (over the fixed set $\mathcal{N}$) is denoted by $\mathbb{MSC}$. Like for topologies, we do not distinguish isomorphic MSCs.

▶ **Example 4.** Two example MSCs are depicted in Figure 3, both having the ring with five processes as underlying topology (for the moment, we ignore the state labels s_i of processes). The events are the endpoints of message arrows, which represent $\lhd_{\mathsf{msg}}$. Process edges are implicitly given; they connect successive events located on the same (top-down) process line. Finally, the mapping ℓ_{M_1} is illustrated on a few events. ◀

3 Parameterized Communicating Automata

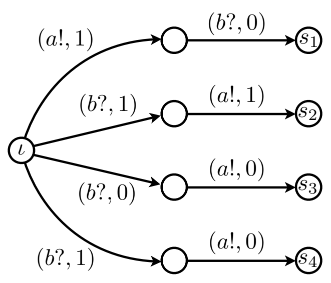
In this section, we introduce our model of a communicating system that can be run on arbitrary topologies of bounded degree.

The idea is that each process of a given topology runs one and the same automaton, whose transitions are labeled with an action of the form $(a!, m)$, which emits a message m through interface a, or $(a?, m)$, which receives m from interface a.

▶ **Definition 5.** A *parameterized communicating automaton (PCA)* over $\mathcal{N}$ is a tuple $\mathcal{A} = (S, \iota, Msg, \Delta, \mathcal{F})$ where S is the finite set of *states*, $\iota \in S$ is the *initial state*, Msg is a nonempty finite set of *messages*, $\Delta \subseteq S \times (\Sigma \times Msg) \times S$ is the *transition relation*, and $\mathcal{F} \in \mathrm{MSO}_t(S)$ is a sentence, representing the acceptance condition.

Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC. A run of $\mathcal{A}$ on M will be a mapping $\rho : E \to S$ satisfying some requirements. Intuitively, $\rho(e)$ is the local state of $\pi(e)$ after executing e. To determine when ρ is a run, we define another mapping, $\rho^- : E \to S$, denoting the source states of a transition: whenever $f \lhd_{\mathsf{proc}} e$, we let $\rho^-(e) = \rho(f)$; moreover, if e is $\lhd_{\mathsf{proc}}$-minimal, we let $\rho^-(e) = \iota$. With this, we say that ρ is a run of $\mathcal{A}$ on M if, for all $(e, f) \in \lhd_{\mathsf{msg}}$, there are $a, b \in \mathcal{N}$ and a message $m \in Msg$ such that $\pi(e) \overset{a\ \ b}{\longmapsto} \pi(f)$, $(\rho^-(e), (a!, m), \rho(e)) \in \Delta$, and $(\rho^-(f), (b?, m), \rho(f)) \in \Delta$. To determine when ρ is accepting, we collect the last states of all processes and define a mapping $\lambda : P \to S$ as follows. Let $p \in P$. If $E_p = \emptyset$, then $\lambda(p) = \iota$; otherwise, $\lambda(p)$ is set to $\rho(e)$ where e is the unique $\lhd_{\mathsf{proc}}$-maximal event of p. Now, run ρ is *accepting* if $(P, \longmapsto, \lambda) \in L(\mathcal{F})$. The set of MSCs that allow for an accepting run is denoted by $L(\mathcal{A})$.

While a run of a PCA is purely operational, it is actually natural to define the acceptance condition in terms of $\mathrm{MSO}_t(S)$, which allows for a global, declarative view of the final configuration. Note that, when we restrict to pipelines, rings, or ranked trees, the acceptance condition could be defined as a finite (tree, respectively) automaton over the alphabet S.



$$\begin{aligned}\mathcal{F} &\equiv \forall u.\lambda(u) \in \{s_1, \dots, s_4\}\\ \mathcal{F}' &\equiv \mathcal{F} \wedge \exists^{=1} u.\lambda(u) = s_1\end{aligned}$$

Figure 4 The PCA $\mathcal{A}'_{\text{token}}$.

▶ **Example 6.** The PCA from Figure 4 describes a simplified version of the IEEE 802.5 token-ring protocol. For illustration, we consider two different acceptance conditions, $\mathcal{F}$ and $\mathcal{F}'$, giving rise to PCAs $\mathcal{A}_{\text{token}}$ and $\mathcal{A}'_{\text{token}}$, respectively. In both cases, a single binary token, which can carry a value from $m \in \{0,1\}$, circulates in a ring. Recall that, in a ring topology, every process has an a-neighbor and a b-neighbor (cf. Figure 2). Initially, the token has value 1. A process that has the token may emit a message and pass it along with the token to its a-neighbor. We will abstract the concrete message away and only consider the token value. Whenever a process receives the token from its b-neighbor, it will forward it to its a-neighbor, while (i) leaving the token value unchanged (the process then ends in state s_2 or s_3), or (ii) changing the token value from 1 to 0, to signal that the message has been received (the process then ends in s_4). Once the process that initially launched the token receives the token with value 0, it goes to state s_1.

Note that the acceptance condition $\mathcal{F}$ of $\mathcal{A}_{\text{token}}$ permits those configurations where all processes terminate in one of the states $s_1, \dots, s_4$. MSC M_1 from Figure 3 depicts an execution of the protocol described above, and we have $M_1 \in L(\mathcal{A}_{\text{token}})$. The state labelings of processes indicate the final local states that are reached in an accepting run. However, one easily verifies that we also have $M_2 \in L(\mathcal{A}_{\text{token}})$, though M_2 should not be considered as an execution of a token-ring protocol: there are two processes that, independently of each other, emit a message/token and end up in s_1. To model the protocol faithfully and rule out such pathological executions, we change the acceptance condition to $\mathcal{F}'$, which adds the requirement that *exactly* one process terminates in s_1. We actually have $M_1 \in L(\mathcal{A}'_{\text{token}})$ and $M_2 \notin L(\mathcal{A}'_{\text{token}})$. ◀

Note that [3, 4] used weaker acceptance conditions, which cannot access the topology. However, Example 6 shows that an acceptance condition given as an MSO$_t$-formula offers some flexibility in modeling parameterized systems. For example, it can be used to simulate several process types [4], the idea being that each process runs a local automaton according to its type. All our results go through in this extended setting. Also note that messages (such as the token value in Example 6) could be made apparent in the MSCs. However, we will always need some "hidden" messages, which are common in communicating automata with fixed topology [10] and significantly extend their expressive power.

Context-Bounded PCAs. Our main results will rely on a restricted version of PCAs, where every process is constrained to execute a bounded number of *contexts*. As discussed in the introduction, contexts come very naturally when modeling distributed protocols. Actually, the behavior of a single process is often divided into a small, or even bounded, number of *rounds*, each describing some restricted communication pattern. Usually, one considers that a round consists of sending a message to each neighbor followed by receiving a message from each neighbor [13]. In this paper, we consider *contexts*, which are somewhat more general than rounds: in a context, one may potentially execute an unbounded number of actions. Moreover, a round can be simulated by a bounded number of contexts. Actually, there exist several natural definitions. A word $w \in \Sigma^*$ is called an

- $(\mathsf{s}{\oplus}\mathsf{r})$-context if $w \in \{a! \mid a \in \mathcal{N}\}^*$ or $w \in \{a? \mid a \in \mathcal{N}\}^*$,
- $(\mathsf{s1}{+}\mathsf{r1})$-context if $w \in \{a!\,, b?\}^*$ for some $a, b \in \mathcal{N}$,
- $(\mathsf{s}{\oplus}\mathsf{r1})$-context if $w \in \{a! \mid a \in \mathcal{N}\}^*$ or $w \in \{b?\}^*$ for some $b \in \mathcal{N}$,
- intf-context if $w \in \{a!\,, a?\}^*$ for some $a \in \mathcal{N}$.

The context type $\mathsf{s1}{\oplus}\mathsf{r}$ ($w \in \{a!\}^*$ for some $a \in \mathcal{N}$ or $w \in \{b? \mid b \in \mathcal{N}\}^*$) is dual to $\mathsf{s}{\oplus}\mathsf{r1}$, and we only consider the latter case. All results for $\mathsf{s}{\oplus}\mathsf{r1}$ in this paper easily transfer to $\mathsf{s1}{\oplus}\mathsf{r}$.

Let $k \geq 1$ be a natural number and $ct \in \{\mathsf{s}{\oplus}\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}, \mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$ be a context type. We say that $w \in \Sigma^*$ is (k, ct)-*bounded* if there are $w_1, \ldots, w_k \in \Sigma^*$ such that $w = w_1 \cdots w_k$ and w_i is a ct-context, for all $i \in [k]$. To lift this definition to MSCs $M = (P, \longmapsto, E, \lhd, \pi)$, we define the projection $M|_p \in \Sigma^*$ of M to a process $p \in P$. Let $e_1 \lhd_{\mathsf{proc}} e_2 \lhd_{\mathsf{proc}} \cdots \lhd_{\mathsf{proc}} e_n$ be the unique process-order preserving enumeration of all events of E_p. We let $M|_p = \ell_M(e_1)\ell_M(e_2)\ldots\ell_M(e_n)$. In particular, $E_p = \emptyset$ implies $M|_p = \varepsilon$. Now, we say that M is (k, ct)-*bounded* if $M|_p$ is (k, ct)-bounded, for all $p \in P$. Let $\mathbb{MSC}_{(k,ct)}$ denote the set of all (k, ct)-bounded MSCs. Given two sets L and L' of MSCs, we write $L \equiv_{(k,ct)} L'$ if $L \cap \mathbb{MSC}_{(k,ct)} = L' \cap \mathbb{MSC}_{(k,ct)}$.

▶ **Example 7.** Consider the PCAs $\mathcal{A}_{\text{token}}$ and $\mathcal{A}'_{\text{token}}$ from Figure 4. Every process executes at most two events so that we have $L(\mathcal{A}'_{\text{token}}) \subseteq L(\mathcal{A}_{\text{token}}) \subseteq \mathbb{MSC}_{(2,ct)}$ for all context types $ct \in \{\mathsf{s}{\oplus}\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}, \mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$. In particular, the MSCs M_1 and M_2 from Figure 3 are $(2, ct)$-bounded.

4 Context-Bounded PCAs are Complementable

Let $ct \in \{\mathsf{s}{\oplus}\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}, \mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$. We say that PCAs are *ct-complementable* if, for every PCA $\mathcal{A}$ and $k \geq 1$, we can effectively construct a PCA $\mathcal{A}'$ such that $L(\mathcal{A}') \equiv_{(k,ct)} \mathbb{MSC} \setminus L(\mathcal{A})$. In general, PCAs are not complementable, and this even holds under certain context bounds.

▶ **Theorem 8.** *Suppose $\mathcal{N} = \{a, b\}$. For all context types $ct \in \{\mathsf{s}{\oplus}\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}\}$, PCAs are* not *ct-complementable.*

The proof uses results from [16, 14]. However, the situation changes when we move to context types $\mathsf{s}{\oplus}\mathsf{r1}$ and intf. We now present the main result of our paper:

▶ **Theorem 9.** *For all $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$, PCAs are ct-complementable.*

The theorem follows directly from a disambiguation construction, which we present as Theorem 10. We call a PCA $\mathcal{A}$ *unambiguous* if, for every MSC M, there is exactly one run (accepting or not) of $\mathcal{A}$ on M. An unambiguous PCA can be easily complemented by negating the acceptance condition.

▶ **Theorem 10.** *Given a PCA $\mathcal{A}$, a natural number $k \geq 1$, and $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$, we can effectively construct an unambiguous PCA $\mathcal{A}'$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L(\mathcal{A}')$.*

The PCA from Figure 4 is not unambiguous, since there are runs of $\mathcal{A}_{\text{token}}$ (or $\mathcal{A}'_{\text{token}}$) on the MSC M_1 from Figure 3 ending, for example, in configurations $s_1 s_2 s_4 s_3 s_3$ or $s_1 s_2 s_2 s_4 s_3$. Unfortunately, a simple power-set construction is not applicable to PCAs, due to the hidden message contents. Note that, in the fixed-topology setting, there is a commonly accepted notion of *deterministic* communicating automata [10], which is different from *unambiguous.* We do not know if Theorem 10 holds for deterministic PCAs.

Proof of Theorem 10

In the remainder of this section, we prove Theorem 10. The proof outline is as follows: We first define an intermediate model of *complete deterministic asynchronous automata (CDAAs).* We will then show that any context-bounded PCA can be converted into a CDAA (Lemma 13) which, in turn, can be converted into an unambiguous PCA (Lemma 12).

▶ **Definition 11.** A *complete deterministic asynchronous automaton (CDAA)* over the set $\mathcal{N}$ is a tuple $\mathcal{B} = (S, \iota, (\delta_{(a,b)})_{(a,b)\,\in\mathcal{N}\times\mathcal{N}}, \mathcal{F})$ where S, ι, and $\mathcal{F}$ are like in PCAs and, for each $(a,b) \in \mathcal{N} \times \mathcal{N}$, we have a (total) function $\delta_{(a,b)} : (S \times S) \to (S \times S)$.

The main motivation behind introducing CDAAs is that, for a given process p, the functions $(\delta_{(a,b)})_{(a,b)\,\in\mathcal{N}\times\mathcal{N}}$ can effectively encode the transitions at each of the neighbors of p. Similarly to PCAs, a run of $\mathcal{B}$ on an MSC $M = (P, \longmapsto, E, \lhd, \pi)$ is a mapping $\rho : E \to S$ such that, for all $(e,f) \in \lhd_{\mathsf{msg}}$, there are $a, b \in \mathcal{N}$ satisfying $\pi(e) \overset{a\ \ b}{\longmapsto} \pi(f)$ and $\delta_{(a,b)}(\rho^-(e), \rho^-(f)) = (\rho(e), \rho(f))$. Whether a run is accepting or not depends on $\mathcal{F}$ and is defined exactly like in PCAs. The set of MSCs that are accepted by $\mathcal{B}$ is denoted by $L(\mathcal{B})$.

▶ **Lemma 12.** *For every CDAA $\mathcal{B}$, there is an unambiguous PCA $\mathcal{A}$ such that $L(\mathcal{B}) = L(\mathcal{A})$.*

Proof. The idea is that the messages of a PCA "guess" the current state of the receiving process. A message can only be received if the guess is correct, so that the resulting PCA is unambiguous. Let $\mathcal{B} = (S, \iota, (\delta_{(a,b)})_{(a,b)\,\in\mathcal{N}\times\mathcal{N}}, \mathcal{F})$ be the given CDAA. We let $\mathcal{A} = (S, \iota, \mathit{Msg}, \Delta, \mathcal{F})$ where $\mathit{Msg} = \mathcal{N} \times \mathcal{N} \times S \times S$ and Δ contains, for every transition $\delta_{(a,b)}(s_1, s_2) = (s'_1, s'_2)$, the tuples $(s_1, a!(a, b, s_1, s_2), s'_1)$ and $(s_2, b?(a, b, s_1, s_2), s'_2)$. Note that $\mathcal{A}$ is indeed unambiguous. Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC and $\rho : E \to S$. From the run definitions, we obtain that ρ is an (accepting) run of $\mathcal{B}$ on M iff it is an (accepting, respectively) run of $\mathcal{A}$ on M. It follows that $L(\mathcal{B}) = L(\mathcal{A})$. ◀

Next, we will describe how an arbitrary context-bounded PCA can be transformed into an equivalent CDAA. This construction is our key technical contribution.

▶ **Lemma 13.** *Let $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$. For every PCA $\mathcal{A}$ and $k \geq 1$, we can effectively construct a CDAA $\mathcal{B}$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L(\mathcal{B})$.*

The remainder of this section is dedicated to the proof of Lemma 13. We do the proof for the more involved case $ct = \mathsf{s}{\oplus}\mathsf{r1}$. Let $\mathcal{A} = (S, \iota, \mathit{Msg}, \Delta, \mathcal{F})$ be a PCA and $k \geq 1$. In the following, we will construct the required CDAA $\mathcal{B} = (S', \iota', (\delta_{(a,b)})_{(a,b)\,\in\mathcal{N}\times\mathcal{N}}, \mathcal{F}')$.

The idea behind our construction is that the current sending process simulates the behavior of all its neighboring receiving processes, storing all possible combinations of global source and target states. In Figure 5, in the beginning, p_2 starts sending to p_3 and p_1. Hence, p_2 keeps track of the local states at p_1 and p_3 as well. This computation spans over what we call a *zone* (the gray-shaded areas in Figure 5). Whenever a sending (receiving) process changes into a receiving (sending, respectively) process, the role of keeping track of the behavior of neighboring processes gets passed on to the new sending process, which results

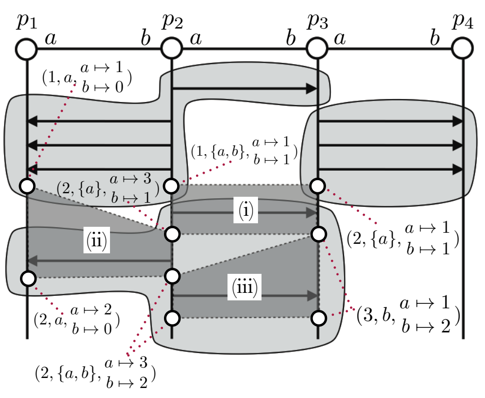


Figure 5 Computing zones in a CDAA $\mathcal{B}$.

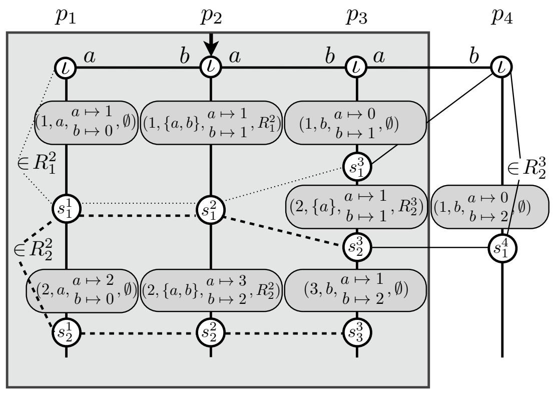


Figure 6 Illustration of $\mathcal{F}' \in \mathrm{MSO}_t(S')$.

in a zone switch. We will see that a bounded number of such changes suffice (Lemma 14). Finally, the acceptance condition $\mathcal{F}'$ checks whether the information stored at each of the processes can be coalesced to get a global run of the given PCA $\mathcal{A}$.

Zones. Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC. An *interval* of M is a (possibly empty) subset of E of the form $\{e_1, e_2, \ldots, e_n\}$ such that $e_1 \lhd_{\mathsf{proc}} e_2 \lhd_{\mathsf{proc}} \ldots \lhd_{\mathsf{proc}} e_n$. A *send context* of M is an interval that consists only of send events. A *receive context* of M is an interval $I \subseteq E$ such that there is $a \in \mathcal{N}$ satisfying $\ell_M(e) = a?$ for all $e \in I$.

A set $\mathcal{Z} \subseteq E$ is called a *zone* of M if there is a nonempty send context I such that the corresponding receive contexts $I_a = \{f \in E \mid e \lhd_{\mathsf{msg}} f$ for some $e \in I$ such that $\ell_M(e) = a!\}$ are intervals for all $a \in \mathcal{N}$, and $\mathcal{Z} = I \cup \bigcup_{a \in \mathcal{N}} I_a$.

Zones help us to maintain the *summary* of a possibly unbounded number of messages in a finite space. By Lemma 14 below, since there is a bound on the number of different zones for each process, the behavior of a PCA can be described succinctly by describing its action on each of the zones.

▶ **Lemma 14.** *[cf. [4]] Let $M = (P, \longmapsto, E, \lhd, \pi)$ be a $(k, \mathsf{s}{\oplus}\mathsf{r}1)$-bounded MSC. There is a partitioning of the events of M into zones such that, for each process $p \in P$, the events of E_p belong to at most $K := k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$ different zones.*

A CDAA that Computes Zones. We now construct a CDAA that, when running on a $(k, \mathsf{s}{\oplus}\mathsf{r}1)$-bounded MSC, computes a "greedy" zone partitioning, for which the bound K from Lemma 14 applies. We explain the intuition by means of Figure 5, which depicts an MSC along with a partitioning of events into different zones. The crucial point for processes is to recognize when to switch to a new zone. Towards this end, a *summary* of the zone is maintained. Each process stores its zone number together with the zone number of its neighboring receiving processes. A sending (receiving) process enters a new zone if the stored zone number of a neighbor does not match the actual zone number of the corresponding neighboring receiving (sending, respectively) process.

In Figure 5, the zone number of p_3 in p_2's first zone is 1. However, at the time of sending the second message from p_2 to p_3, the zone number of p_3 is 2 which does not match the information stored with p_2. This prompts p_2 to define a new zone and update the zone number of p_3.

A *sending process enters a new zone* when (a) it was a receiving process earlier, or (b) the zone number of a receiving process does not match. Similarly, a *receiving process enters a new zone* when (a) it was a sending process earlier, or (b) it was receiving previously from a different process, or (c) the zone number of the sending process does not match. This is formally defined in Equations (1) and (2) below.

We now formally describe the CDAA $\mathcal{B}$. A *zone state* is a tuple (i, τ, κ, R) where

- $i \in \{0, \ldots, K\}$ is the current zone number, which indicates that a process traverses its i-th zone (or, equivalently, has switched to a new zone $i - 1$ times),
- $\tau \in 2^{\mathcal{N}} \cup \mathcal{N}$ denotes the role of a process in the current zone (if $\tau \subseteq \mathcal{N}$, it has been sending through the interfaces in τ; if $\tau \in \mathcal{N}$, it is receiving from τ),
- $\kappa : \mathcal{N} \to \{0, \ldots, K\}$ denotes the knowledge about each neighbor, and
- $R \subseteq (S^{\mathcal{N} \cup \{\mathsf{self}\}})^2$ is the set of possible global steps that the zone may induce; each step involves a source and target state for the current process as well as its neighbors. As the sending process simulates the receivers' steps, we let $R = \emptyset$ whenever $\tau \in \mathcal{N}$.

Let Z be the set of zone states. For $(a, b) \in \mathcal{N} \times \mathcal{N}$, we define a *partial* "update" function $\delta^{\text{zone}}_{(a,b)} : (Z \times Z) \rightharpoonup (Z \times Z)$ by

$$\delta^{\text{zone}}_{(a,b)}((i_1, \tau_1, \kappa_1, R_1), (i_2, \tau_2, \kappa_2, R_2)) = ((i'_1, \tau'_1, \kappa_1[a \mapsto i'_2], R'_1), (i'_2, b, \kappa_2[b \mapsto i'_1], \emptyset))$$

where

$$i'_1 = \begin{cases} i_1 + 1 & \text{if } i_1 = 0 \ \text{ or } \ \tau_1 \in \mathcal{N} \ \text{ or } \ (a \in \tau_1 \ \text{ and } \ \kappa_1(a) \neq i_2) \\ i_1 & \text{otherwise} \end{cases} \tag{1}$$

$$i'_2 = \begin{cases} i_2 + 1 & \text{if } \tau_2 \neq b \ \text{ or } \ \kappa_2(b) \neq i_1 \\ i_2 & \text{otherwise} \end{cases} \tag{2}$$

$$\tau'_1 = \begin{cases} \{a\} & \text{if } i'_1 = i_1 + 1 \\ \tau_1 \cup \{a\} & \text{otherwise} \end{cases} \qquad R'_1 = \begin{cases} R & \text{if } i'_1 = i_1 + 1 \\ R_1 \circ R & \text{otherwise} \end{cases}$$

with R being the set of pairs $(\bar{s}, \bar{s}') \in (S^{\mathcal{N} \cup \{\mathsf{self}\}})^2$ such that there is $m \in \mathit{Msg}$ with $(\bar{s}_{\mathsf{self}}, (a!, m), \bar{s}'_{\mathsf{self}}) \in \Delta$, $(\bar{s}_a, (b?, m), \bar{s}'_a) \in \Delta$, and $\bar{s}_c = \bar{s}'_c$ for all $c \in \mathcal{N} \setminus \{a\}$.

The function $\delta^{\text{zone}}_{(a,b)}$ is illustrated in Figure 5 (omitting the R-component) for the three different cases that can occur: (i) both processes increase their zone number; (ii) only the receiver increases its zone number; (iii) none of the processes increases its zone number.

A state of $\mathcal{B}$ is a sequence of zone states, so that a process can keep track of the zones that it traverses. Formally, we let S' be the set of words over Z of the form $(0, \emptyset, \kappa_0, \emptyset)(1, \tau_1, \kappa_1, R_1) \ldots (n, \tau_n, \kappa_n, R_n)$ where $n \in \{0, \ldots, K\}$ and $\kappa_0(a) = 0$ for all $a \in \mathcal{N}$. The initial state is $\iota' = (0, \emptyset, \kappa_0, \emptyset)$. Actually, S' will also contain a distinguished sink state, as explained below. Note that the size of S' is exponential in K (and, therefore, in k).

We are now ready to define the transition function $\delta_{(a,b)} : (S' \times S') \to (S' \times S')$. Essentially, we take $\delta^{\text{zone}}_{(a,b)}$, but we append a new zone state when the zone number is increased. Let $z_1 = (i_1, \tau_1, \kappa_1, R_1) \in Z$ and $z_2 = (i_2, \tau_2, \kappa_2, R_2) \in Z$. Moreover, suppose $\delta^{\text{zone}}_{(a,b)}(z_1, z_2) = (z'_1, z'_2)$ where $z'_1 = (i'_1, \tau'_1, \kappa'_1, R'_1)$ and $z'_2 = (i'_2, \tau'_2, \kappa'_2, R'_2)$. Then, we let

$$\delta_{(a,b)}(w_1 z_1\, ,\, w_2 z_2) = \begin{cases} (w_1 z'_1\, ,\, w_2 z'_2) & \text{if } i'_1 = i_1 \ \text{ and } \ i'_2 = i_2 \\ (w_1 z'_1\, ,\, w_2 z_2 z'_2) & \text{if } i'_1 = i_1 \ \text{ and } \ i'_2 = i_2 + 1 \\ (w_1 z_1 z'_1\, ,\, w_2 z_2 z'_2) & \text{if } i'_1 = i_1 + 1 \ \text{ and } \ i'_2 = i_2 + 1 \end{cases}$$

Note that the case $i'_1 = i_1 + 1 \wedge i'_2 = i_2$ can actually never happen. Nonetheless, $\delta_{(a,b)}$ is still a partial function. However, adding a sink state, we easily obtain a function that is complete.

The Acceptance Condition. It remains to determine the acceptance condition of $\mathcal{B}$. The formula $\mathcal{F}' \in \mathrm{MSO}_t(S')$ will check whether there is a concrete choice of local states that is consistent with the zone abstraction and, in particular, with the relations R collected during that run in the zone states. Let T be the set of sequences of the form $\iota s_1 \dots s_n$ where $n \in \{0, \dots, K\}$ and $s_i \in S$ for all i. The idea is that s_i is the local state that a process reaches *after* traversing its i-th zone. The formula will now guess such a sequence for every process and check if this choice matches the abstract run. To verify if the local states correspond to the relation R stored in some constituent sending process p, it is sufficient to look at the adjacent neighbors of p.

This is illustrated in Figure 6 for the zone abstraction from Figure 5. Process p_2, for example, stores both the relations R_1^2 and R_2^2, and we have to check if this corresponds to the sequences from T that the formula had guessed for every process (the white circles). To do so, it is indeed enough to look at the neighborhood of p_2, which is highlighted in gray. The guess is accepted only if the state at the beginning of a zone matches the state at the end of the previous zone. For example, in Figure 6, the formula collects the pair of tuples $((\iota, \iota, \iota), (s_1^1, s_1^2, s_1^3))$ and verifies if it is contained in R_1^2. Also, it collects the pair $((s_1^1, s_1^2, s_2^3), (s_2^1, s_2^2, s_3^3))$ and checks if it is contained in R_2^2. Similarly, looking at the neighborhood of p_3, it verifies whether $((s_1^2, s_1^3, \iota), (s_1^2, s_2^3, s_1^4)) \in R_2^3$.

Let us be more precise. Suppose the final configuration reached by $\mathcal{B}$ is $(P, \longmapsto, \lambda')$ with $\lambda' : P \to S'$. By means of second-order variables U_t, with t ranging over T, the formula $\mathcal{F}'$ guesses an assignment $\sigma : P \to T$. It will then check that, for all $p \in P$ with, say, $\lambda'(p) = \iota'(1, \tau_1, \kappa_1, R_1) \dots (n_p, \tau_{n_p}, \kappa_{n_p}, R_{n_p}) \in S'$, the following hold:

- the sequence $\sigma(p)$ is of the form $s_0 s_1 \dots s_{n_p}$ (in the following, we let $\sigma(p)_i$ refer to s_i),
- for all $i \in [n_p]$ with $\tau_i \subseteq \mathcal{N}$, there is $(\bar{s}, \bar{s}') \in R_i$ such that (i) $\bar{s}_{\mathsf{self}} = s_{i-1}$ and $\bar{s}'_{\mathsf{self}} = s_i$, and (ii) for all $p \xmapsto{a\ \ b} q$ such that $a \in \tau_i$, we have $\bar{s}_a = \sigma(q)_{\kappa_i(a)-1}$ and $\bar{s}'_a = \sigma(q)_{\kappa_i(a)}$.

These requirements can be expressed in $\mathrm{MSO}_t(S')$. Finally, to incorporate the acceptance condition $\mathcal{F} \in \mathrm{MSO}_t(S)$, we simply replace an atomic formula $\lambda(u) = s$, where $s \in S$, by the disjunction of all formulas $u \in U_t$ such that $t \in T$ ends in s. This concludes the construction of the CDAA $\mathcal{B}$.

5 Monadic Second-Order Logic

MSO logic over MSCs is two-sorted, as it shall reason about processes and events. By $u, v, w, \dots$ and $U, V, W, \dots$, we denote first-order and second-order variables, which range over processes and sets of processes, respectively. Moreover, by $x, y, z, \dots$ and $X, Y, Z, \dots$, we denote variables ranging over (sets of, respectively) events. The logic MSO_m is given by the grammar $\varphi ::= u \xmapsto{a\ \ b} v \mid u = v \mid u \in U \mid \exists u.\varphi \mid \exists U.\varphi \mid \neg\varphi \mid \varphi \vee \varphi \mid x \lhd_{\mathsf{proc}} y \mid x \lhd_{\mathsf{msg}} y \mid x = y \mid x@u \mid x \in X \mid \exists x.\varphi \mid \exists X.\varphi$ where $a, b \in \mathcal{N}$.

MSO_m formulas are interpreted over MSCs $M = (P, \longmapsto, E, \lhd, \pi)$. Hereby, free variables u and x are interpreted by a function $\mathcal{I}$ as a process $\mathcal{I}(u) \in P$ and an event $\mathcal{I}(x) \in E$, respectively. Similarly, U and X are interpreted as sets. We write $M, \mathcal{I} \models u \xmapsto{a\ \ b} v$ if $\mathcal{I}(u) \xmapsto{a\ \ b} \mathcal{I}(v)$ and $M, \mathcal{I} \models x@u$ if $\pi(\mathcal{I}(x)) = \mathcal{I}(u)$. Thus, $x@u$ says that "x is located at u". The semantics of other formulas is as expected. When φ is a sentence, i.e., a formula without free variables, then its truth value is independent of an interpretation function so that we can simply write $M \models \varphi$ instead of $M, \mathcal{I} \models \varphi$. The set of MSCs M such that $M \models \varphi$ is denoted by $L(\varphi)$.

▶ **Example 15.** Let us resume the token-ring protocol from Example 6. We would like to express that there is a process that emits a message and gets an acknowledgment that results from a sequence of forwards through interface a. We first let $\mathrm{fwd}(x,y) \equiv x \xmapsto{a\ \ b} y \wedge \exists z.(x \lhd_{\mathsf{proc}} z \lhd_{\mathsf{msg}} y)$ where $x \xmapsto{a\ \ b} y$ is a shorthand for $\exists u.\exists v.(x@u \wedge y@v \wedge u \xmapsto{a\ \ b} v)$. It is well known that the transitive closure of the relation induced by $\mathrm{fwd}(x,y)$ is definable in MSO_m-logic, too. Let $\mathrm{fwd}^+(x,y)$ be the corresponding formula. It expresses that there is a sequence of events leading from x to y that alternatingly takes process and message edges, hereby following the causal order. With this, the desired formula is $\varphi \equiv \exists x,y,z.(x \lhd_{\mathsf{proc}} y \wedge x \lhd_{\mathsf{msg}} z \wedge x \xmapsto{a\ \ b} z \wedge \mathrm{fwd}^+(z,y)) \in \mathrm{MSO}_m$. Consider Figures 3 and 4. We have $M_1 \models \varphi$ and $M_2 \not\models \varphi$, as well as $L(\mathcal{A}'_{\text{token}}) \subseteq L(\varphi)$. ◀

▶ **Theorem 16.** *Let $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$, $k \geq 1$, and $L \subseteq \mathbb{MSC}$. There is a PCA $\mathcal{A}$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L$ iff there is a sentence $\varphi \in \mathrm{MSO}_m$ such that $L(\varphi) \equiv_{(k,ct)} L$.*

The direction "$\Longrightarrow$" follows a standard pattern and is actually independent of a context bound. For the direction "$\Longleftarrow$", we proceed by induction, crucially relying on Theorem 9. Note that there are some subtleties in the translation, which arise from the fact that MSO_m mixes event and process variables.

By the results from [4], we obtain decidability of MSO_m model checking as a corollary.

▶ **Theorem 17.** *Let $\mathfrak{T}$ be one of the following: the class of rings, the class of pipelines, or the class of ranked trees. The following problem is decidable, for all $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$:*

Input:	*A PCA $\mathcal{A}$, a sentence $\varphi \in \mathrm{MSO}_m$, and $k \geq 1$.*
Question:	*Do we have $M \models \varphi$ for all MSCs $M = (P, \longmapsto, E, \lhd, \pi) \in L(\mathcal{A}) \cap \mathbb{MSC}_{(k,ct)}$ such that $(P, \longmapsto) \in \mathfrak{T}$?*

Note that MSO_m is a powerful logic and it may actually be used for the verification of extended models that involve registers to store process identities (pids). MSO logic is able to trace back the origin of a pid so that an additional equality predicate on pids can be reduced to an MSO formula over a finite alphabet. This would allow us to model and verify leader-election protocols. It will be worthwhile to explore this in future work.

References

1 P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI'13*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013.

2 B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.

3 B. Bollig. Logic for communicating automata with parameterized topology. In *CSL-LICS'14*. ACM Press, 2014.

4 B. Bollig, P. Gastin, and J. Schubert. Parameterized Verification of Communicating Automata under Context Bounds. In *RP'14*, volume 8762 of *LNCS*, pages 45–57, 2014.

5 D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2), 1983.

6 G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FoSSaCS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.

7 E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.

8 J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, volume 25 of *LIPIcs*, pages 1–10, 2014.

9 J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, volume 8044 of *LNCS*, pages 124–140. Springer, 2013.

10 B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundam. Inform.*, 80(1-3):147–167, 2007.

11 S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. In *LATIN'10*, volume 6034 of *LNCS*, pages 96–107. Springer, 2010.

12 S. La Torre, M. Napoli, and G. Parlato. Scope-bounded pushdown languages. In *DLT'14*, volume 8633 of *LNCS*, pages 116–128. Springer, 2014.

13 N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., 1996.

14 O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Information and Computation*, 179(2):356–383, 2002.

15 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

16 W. Thomas. Elements of an automata theory over partial orders. In *POMIV'96*, volume 29 of *DIMACS*. AMS, 1996.

Distributed Synthesis for Acyclic Architectures

Anca Muscholl[1] **and Igor Walukiewicz**[2]

1 **Université de Bordeaux, France**
2 **CNRS, Université de Bordeaux, France**

Abstract

The distributed synthesis problem is about constructing correct distributed systems, i.e., systems that satisfy a given specification. We consider a slightly more general problem of distributed control, where the goal is to restrict the behavior of a given distributed system in order to satisfy the specification. Our systems are finite state machines that communicate via rendez-vous (Zielonka automata). We show decidability of the synthesis problem for all ω-regular local specifications, under the restriction that the communication graph of the system is acyclic. This result extends a previous decidability result for a restricted form of local reachability specifications.

1998 ACM Subject Classification D.2.4 [Software Engineering] Software/Program Verification, F.3.1 [Logics and Meanings of Programs] Specifying and Verifying and Reasoning about Programs

Keywords and phrases Distributed synthesis, distributed control, causal memory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.639

1 Introduction

Synthesizing distributed systems from specifications is an attractive objective, since distributed systems are notoriously difficult to get right. Unfortunately, there are very few known decidable frameworks for distributed synthesis. We study a framework for synthesis of open systems that is based on rendez-vous communication and causal memory. In particular, causal memory implies that although the specification can say when a communication takes place, it cannot limit the information that is transmitted during communication. This choice is both realistic and avoids some pathological reasons for undecidability. We show a decidability result for acyclic communication graphs and local ω-regular specifications.

Instead of synthesis we actually work in the more general framework of distributed control. Our setting is a direct adaptation of the supervisory control framework of Ramadge and Wonham [17]. In this framework we are given a plant (a finite automaton) where some of the actions are uncontrollable, and a specification. The goal is to construct a controller (another finite automaton) such that its product with the plant satisfies the specification. The controller is not allowed to block uncontrollable actions, in other words, in every state there is a transition on every uncontrollable action. The controlled plant has less behaviours, resulting from restricting controllable actions of the plant. In our case the formulation is exactly the same, but we consider Zielonka automata instead of finite automata, as plants and controllers. Considering parallel devices, such as Zielonka automata, in the standard definition of control gives an elegant formulation of the distributed control problem.

Zielonka automata [19, 14] are by now a well-established model of distributed computation. Such a device is an asynchronous product of finite-state processes synchronizing on shared actions. Asynchronicity means that processes can progress at different speed. The synchronization on shared actions allows the synchronizing processes to exchange information, in particular the controllers can transfer control information with each synchronization. This model can encode some common synchronization primitives available in modern multi-core processors for implementing concurrent data structures, like compare-and-swap.

We show decidability of the control problem for Zielonka automata where the communication graph is acyclic: a process can communicate (synchronize) with its parent and its children. Our specifications are conjunctions of ω-regular specifications, one for each of the component processes. We allow uncontrollable communication actions – the only restriction is that all communication actions must be binary. Uncontrollable communications add expressive power to the setting, for instance it is possible to model asymmetric situations where communication can be refused by one partner, but not by the other one.

Our result extends [7] that showed decidability for a restricted form of local reachability objectives (blocking final states). We still get the same complexity as in [7]: non-elementary in general, and EXPTIME for architectures of depth 1. The extension to all ω-regular objectives allows to express fairness constraints but at the same time introduces important technical obstacles. Indeed, for our construction to work it is essential that we enrich the framework by uncontrollable synchronization actions. This makes a separation into controllable and uncontrollable states impossible. In consequence, we are led to abandon the game metaphor, to invent new arguments, and to design a new proof structure.

Most research on distributed synthesis and control has been done in the setting proposed by Pnueli and Rosner [16]. This setting is also based on shared-variable communication, however the controllers there are not free to pass additional information between processes. So the latter model leads to partial information games, and decidability of synthesis holds only for variants of pipelines [10, 11, 4]. While specifications leading to undecidability are very artificial, no elegant solution to eliminate them exists at present. The synthesis setting is investigated in [11] for local specifications, meaning that each process has its own, linear-time specification. More relaxed variants of synthesis have been proposed, where the specification does not fully describe the communication of the synthesized system. One approach consists in adding communication in order to combine local knowledge, as proposed for example in [8]. Another approach is to use specifications only for describing external communication, as done in [6] on strongly connected architectures where processes communicate via signals.

Apart from [7], three related decidability results for synthesis with causal memory are known. The first one [5] restricts the alphabet of actions: control with reachability condition is decidable for co-graph alphabets. This restriction excludes client-server architectures, which are captured by our setting. The second result [12] shows decidability by restricting the plant: roughly speaking, the restriction says that every process can have only bounded missing knowledge about the other processes, unless they diverge (see also [15] that shows a doubly exponential upper bound). The proof of [12] goes beyond the controller synthesis problem, by coding it into monadic second-order theory of event structures and showing that this theory is decidable when the criterion on the plant holds. Unfortunately, very simple plants have a decidable control problem and at the same time an undecidable MSO-theory. Safety games on Petri nets are considered in [3], where decidability in EXPTIME is shown in the case when there is a single environment player. Note that the same complexity is achieved in our model with client-server architectures and no restriction on the environment. Game semantics and asynchronous games played on event structures are introduced in [13]. Such games are investigated in [9] from a game-theoretical viewpoint, showing a Borel determinacy result under some restrictions.

Overview. In Section 2 we state and motivate our control problem. In Section 3 we give the main lines of the proof. A complete version of the paper is available at hal-00946554.

2 Control for Zielonka automata

In this section we introduce our control problem for Zielonka automata, adapting the definition of supervisory control [17] to our model.

A Zielonka automaton [19, 14] is a simple distributed finite-state device. Such an automaton is a parallel composition of several finite automata, called *processes*, synchronizing on shared actions. There is no global clock, so between two synchronizations, two processes can do a different number of actions. Because of this, Zielonka automata are also called asynchronous automata.

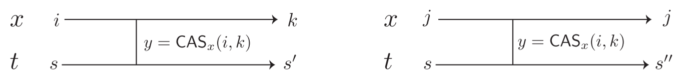
A *distributed action alphabet* on a finite set $\mathbb{P}$ of processes is a pair (Σ, dom), where Σ is a finite set of *actions* and $dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)$ is a *location function.* The location $dom(a)$ of action $a \in \Sigma$ comprises all processes that need to synchronize in order to perform this action. Actions from $\Sigma_p = \{a \in \Sigma \mid p \in dom(a)\}$ are called *p-actions.* We write $\Sigma_p^{loc} = \{a \mid dom(a) = \{p\}\}$ for the set of *local* actions of p.

A (deterministic) *Zielonka automaton* $\mathcal{A} = \langle \{S_p\}_{p \in \mathbb{P}}, s_{in}, \{\delta_a\}_{a \in \Sigma}\rangle$ is given by:

- for every process p a finite set S_p of (local) states,
- the initial state $s_{in} \in \prod_{p \in \mathbb{P}} S_p$,
- for every action $a \in \Sigma$ a partial transition function $\delta_a : \prod_{p \in dom(a)} S_p \dot{\to} \prod_{p \in dom(a)} S_p$ on tuples of states of processes in $dom(a)$.

▶ **Example 1.** Boolean multi-threaded programs with shared variables can be modelled as Zielonka automata. As an example we describe the translation for the *compare-and-swap* (CAS) instruction. This instruction has 3 parameters: $\mathsf{CAS}(x\colon \mathsf{variable};\ old, new\colon \mathsf{int})$. Its effect is to return the value of x and at the same time set the value of x to *new*, but only if the previous value of x was equal to *old*. The compare-and-swap operation is a widely used primitive in implementations of concurrent data structures, and has hardware support in most contemporary multiprocessor architectures.

Suppose that we have a thread t, and a shared variable x that is accessed by a CAS operation in t via $y := \mathsf{CAS}_x(i, k)$. So y is a local variable of t. In the Zielonka automaton we will have one process modelling thread t, and one process for variable x. The states of t will be valuations of local variables. The states of x will be the values x can take. The CAS instruction above becomes a synchronization action. We have the following two types of transitions on this action:



Notice that in state s', we have $y = i$, whereas in s'', we have $y = j$.

For convenience, we abbreviate a tuple $(s_p)_{p \in P}$ of local states by s_P, where $P \subseteq \mathbb{P}$. We also talk about S_p as the set of *p-states.*

A Zielonka automaton can be seen as a sequential automaton with the state set $S = \prod_{p \in \mathbb{P}} S_p$ and transitions $s \xrightarrow{a} s'$ if $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a$, and $s_{\mathbb{P} \setminus dom(a)} = s'_{\mathbb{P} \setminus dom(a)}$. So the states of this automaton are the tuples of states of the processes of the Zielonka automaton. For a process p we will talk about the *p-component* of the state. A run of $\mathcal{A}$ is a finite or infinite sequence of transitions starting in s_{in}. Since the automaton is deterministic, a run is determined by the sequence of labels of the transitions. We will write $run(u)$ for the run determined by the sequence $u \in \Sigma^\infty$. Observe that $run(u)$ may be undefined since the transition function of $\mathcal{A}$ is partial. We will also talk about the projection of the run on component p, denoted $run_p(u)$, that is the projection on component p of the subsequence of the run containing the transitions involving p. We will assume that every local state of $\mathcal{A}$ occurs in some run. By $dom(u)$ we denote the union of $dom(a)$, for all $a \in \Sigma$ occurring in u.

We will be interested in maximal runs of Zielonka automata. For parallel devices the notion of a maximal run is not that evident, as one may want to impose some fairness conditions. We settle here for a minimal sensible fairness requirement. It says that a run is maximal if processes who have only finitely many actions in the run cannot perform any additional action.

▶ **Definition 2** (Maximal run). For a word $w \in \Sigma^\infty$ such that $run(w)$ is defined, we say that $run(w)$ is *maximal* if there is no decomposition $w = uv$, and no action $a \in \Sigma$ such that $dom(v) \cap dom(a) = \emptyset$ and $run(uav)$ is defined.

Automata can be equipped with a *correctness condition.* We prefer to talk about correctness condition rather than acceptance condition since we will be interested in the set of runs of an automaton rather than in the set of words it accepts. We will consider local regular correctness conditions: every process has its own correctness condition $Corr_p$. A run of $\mathcal{A}$ is *correct* if for every process p, the projection of the run on the transitions of $\mathcal{A}_p$ is in $Corr_p$. Condition $Corr_p$ is specified by a set $T_p \subseteq S_p$ of terminal states and an ω-regular set $\Omega_p \subseteq (S_p \times \Sigma_p \times S_p)^\omega$. A sequence $(s_p^0, a_0, s_p^1)(s_p^1, a_1, s_p^2)\dots$ satisfies $Corr_p$ if either: (i) it is finite and ends with a state from T_p, or (ii) it is infinite and belongs to Ω_p. At this stage the set of terminal states T_p may look unnecessary, but it will simplify our constructions later.

Finally, we will need the notion of *synchronized product* $\mathcal{A} \times \mathcal{C}$ of two Zielonka automata. For $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a^A\}_{a\in\Sigma}\rangle$ and $\mathcal{C} = \langle \{C_p\}_{p\in\mathbb{P}}, c_{in}, \{\delta_a^C\}_{a\in\Sigma}\rangle$ let $\mathcal{A} \times \mathcal{C} = \langle \{S_p \times C_p\}_{p\in\mathbb{P}}, (s_{in}, c_{in}), \{\delta_a^\times)_{a\in\Sigma}\}\rangle$ where there is a transition from $(s_{dom(a)}, c_{dom(a)})$ to $(s'_{dom(a)}, c'_{dom(a)})$ in $\delta_a^\times$ iff $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a^A$ and $(c_{dom(a)}, c'_{dom(a)}) \in \delta_a^C$.

To define the control problem for Zielonka automata we fix a distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^\mathbb{P} \setminus \emptyset)\rangle$. We partition Σ into the set of *system actions* Σ^{sys} and *environment actions* Σ^{env}. Below we will introduce the notion of controller, and require that it does not block environment actions. For this reason we speak about *controllable/uncontrollable* actions when referring to system/environment actions. We impose three simplifying assumptions: (1) All actions are at most binary ($|dom(a)| \le 2$ for every $a \in \Sigma$); (2) every process has some controllable action; (3) all controllable actions are local. Among the three conditions only the first one is indeed a restriction of our setting. The other two are not true limitations, in particular controllable shared actions can be simulated by a local controllable choice, followed by non-controllable local or shared actions (see full version of the paper)

▶ **Definition 3** (Controller, Correct Controller). A *controller* is a Zielonka automaton that cannot block environment (uncontrollable) actions. In other words, from every state every environment action is possible: for every $b \in \Sigma^{env}$, δ_b is a total function. We say that a controller $\mathcal{C}$ *is correct for* a plant $\mathcal{A}$ if all maximal runs of $\mathcal{A} \times \mathcal{C}$ satisfy the correctness condition of $\mathcal{A}$.

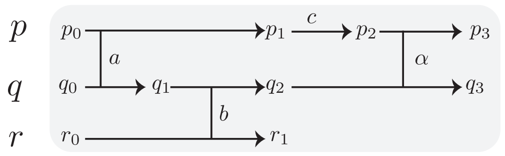
Recall that an action is possible in $\mathcal{A} \times \mathcal{C}$ iff it is possible in both $\mathcal{A}$ and $\mathcal{C}$. By the above definition, environment actions are always possible in $\mathcal{C}$. The major difference between the controlled system $\mathcal{A} \times \mathcal{C}$ and and $\mathcal{A}$ is that the states of $\mathcal{A} \times \mathcal{C}$ carry the additional information computed by $\mathcal{C}$, and that $\mathcal{A}\times\mathcal{C}$ may have less behaviors, resulting from disallowing controllable actions by $\mathcal{C}$.

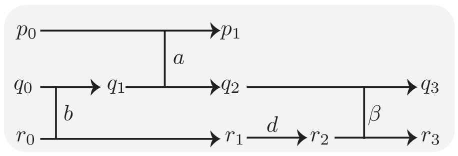
The correctness of $\mathcal{C}$ means that all the runs of $\mathcal{A}$ that are *allowed* by $\mathcal{C}$ are correct. In particular, $\mathcal{C}$ does not have a correctness condition by itself. Considering only maximal runs of $\mathcal{A} \times \mathcal{C}$ imposes some minimal fairness conditions: for example it implies that if a process can do a local action almost always, then it will eventually do some action.

▶ **Definition 4** (Control problem)**.** Given a distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)\rangle$ together with a partition of actions $(\Sigma^{sys}, \Sigma^{env})$, and given a Zielonka automaton $\mathcal{A}$ over this alphabet, find a controller $\mathcal{C}$ over the same alphabet such that $\mathcal{C}$ is correct for $\mathcal{A}$.

The important point in our definition is that the controller has the same distributed alphabet as the automaton it controls, in other words the controller is not allowed to introduce additional synchronizations between processes.

▶ **Example 5.** We give an example showing how causal memory works and helps to construct controllers. Consider an automaton $\mathcal{A}$ with 3 processes: p, q, r. We would like to control it so that the only two possible runs of $\mathcal{A}$ are the following:





So p and q should synchronize on α when action a happened before b, otherwise q and r should synchronize on β. Communication actions are uncontrollable, but the transitions of $\mathcal{A}$ are such that there are local controllable actions c and d that enable communication on α and β respectively. So the controller should block either c or d depending on the order between a and b. The transitions of $\mathcal{A}$ are as follows:

$$\begin{array}{cccc}
\delta_a(p_0,q_0) = (p_1,q_1) & \delta_a(p_0,q_1) = (p_1,q_2) & \delta_b(q_0,r_0) = (q_1,r_1) & \delta_b(q_1,r_0) = (q_2,r_1) \\
\delta_c(p_1) = p_2 & \delta_\alpha(p_2,q_2) = (p_3,q_3) & \delta_d(r_1) = r_2 & \delta_\beta(q_2,r_2) = (q_3,r_3)
\end{array}$$

These transitions allow the two behaviors depicted above but also two unwanted ones, as say, when a happens before b and then we see β. Note that the specification of desired behaviors is a local condition on q. So by encoding some information in states of q this condition can be expressed by a set of terminal states T_q. We will not do this for readability.

The controller $\mathcal{C}$ for $\mathcal{A}$ will mimic the structure of $\mathcal{A}$: for every state of $\mathcal{A}$ there will be in $\mathcal{C}$ a state with over-line. So, for example, the states of q in $\mathcal{C}$ will be $\overline{q}_0, \dots, \overline{q}_3$. Moreover $\mathcal{C}$ will have two new states $\underline{p_1}$ and $\underline{r_1}$. The transitions will be

$$\begin{array}{cccc}
\delta_a(\overline{p}_0,\overline{q}_0) = (\overline{p}_1,\overline{q}_1) & \delta_a(\overline{p}_0,\overline{q}_1) = (\underline{p_1},\overline{q}_2) & \delta_b(\overline{q}_0,\overline{r}_0) = (\overline{q}_1,\overline{r}_1) & \delta_b(\overline{q}_1,\overline{r}_0) = (\overline{q}_2,\underline{r_1}) \\
\delta_c(\overline{p}_1) = \overline{p}_2 & \delta_c(\underline{p_1}) = \perp & \delta_d(\overline{r}_1) = \overline{r}_2 & \delta_d(\underline{r_1}) = \perp \\
\delta_\alpha(\overline{p}_2,\overline{q}_2) = (\overline{p}_3,\overline{q}_3) & & \delta_\beta(\overline{q}_2,\overline{r}_2) = (\overline{q}_3,\overline{r}_3) &
\end{array}$$

Observe that c is blocked in $\underline{p_1}$, and so is d from $\underline{r_1}$. It is easy to verify that the runs of $\mathcal{A} \times \mathcal{C}$ are as required, so $\mathcal{C}$ is a correct controller for $\mathcal{A}$. (Actually the definition of a controller forces us to make transitions of $\mathcal{C}$ total on uncontrollable actions. We can do it in arbitrary way as this will not add new behaviors to $\mathcal{A} \times \mathcal{C}$.)

This example shows several phenomena. The states of $\mathcal{C}$ are the states of $\mathcal{A}$ coupled with some additional information. We formalize this later under a notion of covering controller. We could also see above a case where a communication is decided by one of the parties. Process p, thanks to a local action, can decide if it wants to communicate via α, but process q has to accept α always. This shows the flexibility given by uncontrollable communication actions. Finally, we could see information passing during communication. In $\mathcal{C}$ process q passes to p and r information about its local state (cf. transitions on a and b).

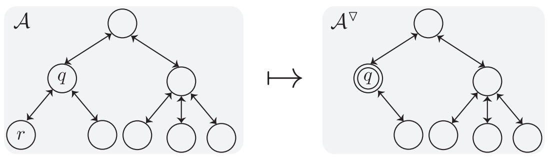


Figure 1 Eliminating process r: r is glued with q.

3 Decidability for acyclic architectures

In this section we present the main result of the paper. We show the decidability of the control problem for Zielonka automata with acyclic architecture. A *communication architecture* of a distributed alphabet is a graph where nodes are processes and edges link processes that have common actions. An *acyclic architecture* is one whose communication graph is acyclic. For example, the communication graph of the alphabet from the example on page 643 is a tree with root q and two successors, p and r.

▶ **Theorem 6.** *The control problem for Zielonka automata over distributed alphabets with acyclic architecture is decidable. If a controller exists, then it can be effectively constructed.*

The remaining of this section is devoted to the outline of the proof of Theorem 6. This proof works by induction on the number $|\mathbb{P}|$ of processes in the automaton. A Zielonka automaton over a single process is just a finite automaton, and the control problem is then just the standard control problem as considered by Ramadge and Wonham but extended to all ω-regular conditions [1]. If there are several processes that do not communicate, then we can solve the problem for each process separately.

Otherwise we choose a leaf process r and its parent q and construct a new plant $\mathcal{A}^\triangledown$ over $\mathbb{P} \setminus \{r\}$. We will show that the control problem for $\mathcal{A}$ has a solution iff the one for $\mathcal{A}^\triangledown$ does. Moreover, for every solution for $\mathcal{A}^\triangledown$ we will be able to construct a solution for $\mathcal{A}$.

For the rest of this section let us fix the distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)\rangle$, the leaf process r and its parent q, and a Zielonka automaton with a correctness condition $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a\}_{a\in\Sigma}, \{Corr_p\}_{p\in\mathbb{P}}\rangle$.

The first step in proving Theorem 6 is to simplify the problem. First, we can restrict to controllers of a special form called covering controllers. Next, we show that the component of $\mathcal{A}$ to be eliminated, that is $\mathcal{A}_r$, can be assumed to have a particular property (r-short). After these preparatory results we will be able to present the reduction of $\mathcal{A}$ to $\mathcal{A}^\triangledown$ (Theorem 13).

3.1 Covering controllers and r-short automata

In this subsection we introduce the notions of covering controllers and r-short automata. We show that in the control problem we can restrict to covering controllers (Lemma 8), and that we can always transform a plant to an r-short one without affecting controllability (Theorem 10).

The notion of a covering controller will simplify the presentation because it will allow us to focus on the runs of the controller instead of a product of the plant and the controller. We have seen already a covering controller in Example 5.

▶ **Definition 7** (Covering controller). Let $\mathcal{C}$ be a Zielonka automaton over the same alphabet as $\mathcal{A}$; let C_p be the set of states of process p in $\mathcal{C}$. Automaton $\mathcal{C}$ is a *covering controller* for $\mathcal{A}$ if there is a function $\pi : \{C_p\}_{p\in\mathbb{P}} \to \{S_p\}_{p\in\mathbb{P}}$, mapping each C_p to S_p and satisfying two conditions: (i) if $c_{dom(b)} \xrightarrow{b} c'_{dom(b)}$ then $\pi(c_{dom(b)}) \xrightarrow{b} \pi(c'_{dom(b)})$; (ii) for every uncontrollable action a: if a is enabled from $\pi(c_{dom(a)})$ then it is also enabled from $c_{dom(a)}$.

▶ Remark. Strictly speaking, a covering controller $\mathcal{C}$ may not be a controller since we do not require that every uncontrollable action is enabled in every state, but only those actions that are enabled in $\mathcal{A}$. From $\mathcal{C}$ one can get a controller $\hat{\mathcal{C}}$ by adding self-loops for all missing uncontrollable transitions.

Notice that thanks to the projection π, a covering controller can inherit the correctness condition of $\mathcal{A}$. Therefore it is enough to look at the runs of $\mathcal{C}$ instead of $\mathcal{A} \times \mathcal{C}$:

▶ **Lemma 8.** *There is a correct controller for $\mathcal{A}$ if and only if there is a covering controller $\mathcal{C}$ for $\mathcal{A}$ such that all the maximal runs of $\mathcal{C}$ satisfy the inherited correctness condition.*

We will refer to a covering controller with the property that all its maximal runs satisfy the inherited correctness condition, as *correct covering controller*.

Now we will focus on making the automaton r-short.

▶ **Definition 9** (r-short)**.** An automaton $\mathcal{A}$ is *r-short* if there is a bound on the number of actions that r can perform without doing a communication with q.

▶ **Theorem 10.** *For every automaton $\mathcal{A}$, we can construct an r-short automaton $\mathcal{A}^{\circledS}$ such that there is a correct controller for $\mathcal{A}$ iff there is one for $\mathcal{A}^{\circledS}$.*

In the rest of this subsection we sketch the proof of this theorem (details are provided in the appendix). This theorem bears some resemblance with the fact that every parity game can be transformed into a finite game: when a loop is closed the winner is decided looking at the ranks on the loop. In order to use a similar construction we must ensure that there is always a controller that is in some sense memoryless on the component r (Lemma 12).

Observe that we can make two simplifying assumptions. First, we assume that the correctness condition on r is a parity condition. That is, it is given by a rank function $\Omega_r : S_r \to \mathbb{N}$ and the set of terminal states T_r. We can assume this since every regular language of infinite sequences can be recognized by a deterministic parity automaton. The second simplification is to assume that the automaton $\mathcal{A}$ is *r-aware* with respect to the parity condition on r. This means that from the state of r one can read the biggest rank that process r has seen the last communication of r with q. It is easy to transform an automaton to an r-aware one.

Given $\mathcal{A}$ we define an r-short automaton $\mathcal{A}^{\circledS}$. All its components will be the same but for the component r. The states $S_r^{\circledS}$ of r will be sequences $w \in S_r^+$ of states of $\mathcal{A}_r$ without repetitions, plus two new states $\top, \bot$. For a local transition $s'_r \xrightarrow{b} s''_r$ in $\mathcal{A}_r$ we have in $\mathcal{A}_r^{\circledS}$ transitions:

$ws'_r \xrightarrow{b} ws'_rs''_r$	if $ws'_rs''_r$ a sequence without repetitions	$(s_q, ws'_r) \xrightarrow{b} (s'_q, s''_r)$
$ws'_r \xrightarrow{b} \top$	if s''_r appears in w and the resulting loop is even	if $(s_q, s'_r) \xrightarrow{b} (s'_q, s''_r)$ in $\mathcal{A}$
$ws'_r \xrightarrow{b} \bot$	if s''_r appears in w and the resulting loop is odd	

To the right we have displayed communication transitions between q and r. Notice that w disappears in communication transitions. The parity condition for $\mathcal{A}^{\circledS}$ is also rather straightforward: it is the same for the components other than r, and for $\mathcal{A}_r$ we have $\Omega^{\circledS}(ws_r) = \Omega(s_r)$, and $T_r^{\circledS} = \{\top\} \cup \{ws_r \mid s_r \in T_r\}$.

It is clear that the length of every sequence of local actions of process r in $\mathcal{A}^{\circledS}$ is bounded by the number of states of $\mathcal{A}_r$.

For the correctness of the reduction we first show how to construct a correct controller $\mathcal{C}$ for $\mathcal{A}$ from a correct controller controller $\mathcal{C}^{\circledS}$ for $\mathcal{A}^{\circledS}$. The idea is that $\mathcal{C}$ simulates $\mathcal{C}^{\circledS}$ but when the execution of the latter gets to $\top$, then the execution detects a loop, so $\mathcal{C}$ can restart $\mathcal{C}^{\circledS}$ from the prefix of the sequence obtained by removing the loop (cf. the definition of $\mathcal{C}^{\circledS}$).

For the other direction, given a correct covering controller $\mathcal{C}$ for $\mathcal{A}$ we show that $\mathcal{C}$ is also a correct controller for $\mathcal{A}^{\circledS}$ provided that $\mathcal{C}$ is r-memoryless as defined below. Then we can conclude by Lemma 12 below, allowing us to assume that $\mathcal{C}$ is memoryless.

Recall that if $\mathcal{C}$ is a covering controller for $\mathcal{A}$ (cf. Definition 7) then there is a function $\pi : \{C_p\}_{p\in\mathbb{P}} \to \{S_p\}_{p\in\mathbb{P}}$, mapping each C_p to S_p and respecting the transition relation: if $c_{dom(b)} \xrightarrow{b} c'_{dom(b)}$ then $\pi(c_{dom(b)}) \xrightarrow{b} \pi(c'_{dom(b)})$.

▶ **Definition 11** (r-memoryless controller)**.** A covering controller $\mathcal{C}$ for $\mathcal{A}$ is *r-memoryless* when for every pair of states $c_r \neq c'_r$ of $\mathcal{C}_r$: if there is a path on local r-actions from c_r to c'_r then $\pi(c_r) \neq \pi(c'_r)$.

Intuitively, a controller can be seen as a strategy, and r-memoryless means that it does not allow the controlled automaton to go twice through the same r-state between two consecutive communication actions of r and q.

▶ **Lemma 12.** *Fix an r-aware automaton $\mathcal{A}$ with a parity correctness condition for process r. If there is a correct controller for $\mathcal{A}$ then there is also one that is covering and r-memoryless.*

The proof of Lemma 12 uses the notion of signatures [18, 2], that is classical in 2-player parity games, for defining a r-memoryless controller $\mathcal{C}^m$ from $\mathcal{C}$. The idea is to use representative states of C_r, defined in each strongly connected component according to a given signature and covering function π.

3.2 The reduced automaton $\mathcal{A}^{\triangledown}$

Equipped with the notions of covering controller and r-short strategy we can now present the construction of the reduced automaton $\mathcal{A}^{\triangledown}$. We suppose that $\mathcal{A} = \langle\{S_p\}_{p\in\mathbb{P}^{\triangledown}}, s_{in}, \{\delta_a\}_{a\in\Sigma}\rangle$ is r-short and we define the reduced automaton $\mathcal{A}^{\triangledown}$ that results by eliminating process r (cf. Figure 1). Let $\mathbb{P}^{\triangledown} = \mathbb{P} \setminus \{r\}$. We construct $\mathcal{A}^{\triangledown} = \langle\{S^{\triangledown}_p\}_{p\in\mathbb{P}^{\triangledown}}, s^{\triangledown}_{in}, \{\delta^{\triangledown}_a\}_{a\in\Sigma^{\triangledown}}\rangle$ where the components are defined below.

All the processes $p \neq q$ of $\mathcal{A}^{\triangledown}$ will be the same as in $\mathcal{A}$. This means: $S^{\triangledown}_p = S_p$, and $\Sigma^{\triangledown}_p = \Sigma_p$. Moreover, all transitions δ_a with $dom(a) \cap \{q, r\} = \emptyset$ are as in $\mathcal{A}$. Finally, in $\mathcal{A}^{\triangledown}$ the correctness condition of $p \neq q$ is the same as in $\mathcal{A}$.

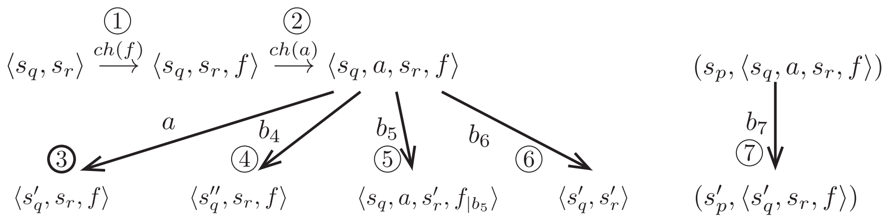
Before defining process q in $\mathcal{A}^{\triangledown}$ let us introduce the notion of r-local strategy. An *r-local strategy from a state $s_r \in S_r$* is a partial function $f : (\Sigma^{loc}_r)^* \to \Sigma^{sys}_r$ mapping sequences from Σ^{loc}_r to actions from Σ^{sys}_r, such that if $f(v) = a$ then $s_r \xrightarrow{va}$ in $\mathcal{A}_r$. Observe that since the automaton $\mathcal{A}$ is r-short, the domain of f is finite.

Given an r-local strategy f from s_r, a local action $a \in \Sigma^{loc}_r$ is *allowed by f* if $f(\epsilon) = a$, or a is uncontrollable. For a allowed by f we denote by $f_{|a}$ the r-local strategy defined by $f_{|a}(v) = f(av)$; this is a strategy from s'_r, where $s_r \xrightarrow{a} s'_r$.

The states of process q in $\mathcal{A}^{\triangledown}$ are of one of the following types:

$$\langle s_q, s_r\rangle\,, \quad \langle s_q, s_r, f\rangle\,, \quad \langle s_q, a, s_r, f\rangle\,,$$

where $s_q \in S_q, s_r \in S_r$, f is a r-local strategy from s_r, and $a \in \Sigma^{loc}_q$. The new initial state for q is $\langle(s_{in})_q, (s_{in})_r\rangle$. Recall that that since $\mathcal{A}$ is r-short, any r-local strategy in $\mathcal{A}_r$ is necessarily finite, so $S^{\triangledown}_q$ is a finite set. Recall also that controllable actions are local.



①
$ch(f)$
②
$ch(a)$
$\langle s_q, s_r\rangle$
$\langle s_q, s_r, f\rangle$
$\langle s_q, a, s_r, f\rangle$
a
b_4
b_5
b_6
③
④
⑤
⑥
$\langle s'_q, s_r, f\rangle$
$\langle s''_q, s_r, f\rangle$
$\langle s_q, a, s'_r, f_{|b_5}\rangle$
$\langle s'_q, s'_r\rangle$
$(s_p, \langle s_q, a, s_r, f\rangle)$
b_7
⑦
$(s'_p, \langle s'_q, s_r, f\rangle)$

Figure 2 Transitions of $\mathcal{A}^{\triangledown}$.

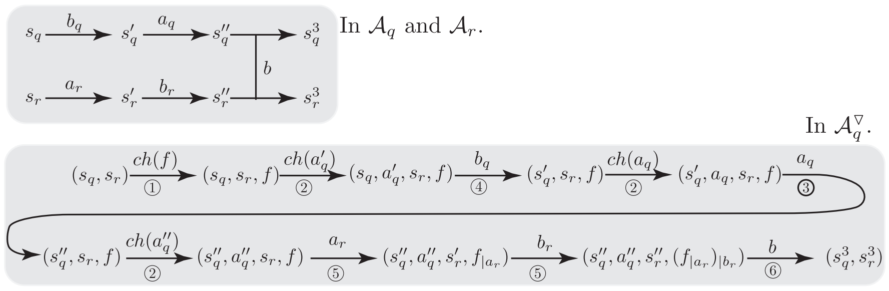


Figure 3 Simulation of $\mathcal{A}_q$ and $\mathcal{A}_r$ by $\mathcal{A}^{\triangledown}_q$.

The transitions of $\mathcal{A}^{\triangledown}_q$ are presented in Figure 2. Transition ① chooses an r-local strategy f. It is followed by transition ② that declares a controllable action $a \in \Sigma^{sys}_q$ that is enabled from s_q. Transition ③ executes the chosen action a; we require $s_q \xrightarrow{a} s'_q$ in $\mathcal{A}_q$. Transition ④ executes an uncontrollable local action $b_4 \in \Sigma^{env}_q$; provided $s_q \xrightarrow{b_4} s''_q$ in $\mathcal{A}_q$. Transition ⑤ executes a local action $b_5 \in \Sigma^{loc}_r$, provided that b_5 is allowed by f and $s_r \xrightarrow{b_5} s'_r$. Transition ⑥ simulates a synchronization on b_6 between q and r; provided $(s_q, s_r) \xrightarrow{b_6} (s'_q, s'_r)$ in $\mathcal{A}$. Finally, transition ⑦ simulates a synchronization between q and $p \neq r$. An example of a simulation of $\mathcal{A}_q$ and $\mathcal{A}_r$ by $\mathcal{A}^{\triangledown}_q$ is presented in Figure 3. The numbers below transitions refer to the corresponding cases from the definition.

To summarize, in $\Sigma^{\triangledown}_q$ we have all actions of Σ_r and Σ_q, but they become uncontrollable. All the new actions of process q in plant $\mathcal{A}^{\triangledown}$ are *controllable*:

- action $ch(f) \in \Sigma^{sys}$, for every local r-strategy f,
- action $ch(a)$, for every $a \in \Sigma^{sys}_q$.

The correctness condition for process q in $\mathcal{A}^{\triangledown}$ is:

1. The correct infinite runs of q in $\mathcal{A}^{\triangledown}$ are those that have the projection on transitions of $\mathcal{A}_q$ correct with respect to $Corr_q$, and either: *(i)* the projection on transitions of $\mathcal{A}_r$ is infinite and correct with respect to $Corr_r$; or *(ii)* the projection on transitions of $\mathcal{A}_r$ is finite and for f, s_r appearing in almost all states of q of the run we have that from s_r all sequences respecting strategy f end in a state from T_r.
2. $T^{\triangledown}_q$ contains states $\langle s_q, s_r, f\rangle$ such that $s_q \in T_q$, and $s_r \in T_r$.

Item 1(ii) in the definition above captures the case where q progresses alone till infinity and blocks r, even though r could reach a terminal state in a couple of moves. Clearly, item 1 can be expressed as an ω-regular condition. The definition of correctness condition is one of the principal places where the r-short assumption is used. Without this assumption we would need to cope with the situation where we have an infinite execution of $\mathcal{A}_q$, and at the same time an infinite execution of $\mathcal{A}_r$ that do not communicate with each other. In this case $\mathcal{A}_q^\triangledown$ would need to fairly simulate both executions in some way.

The reduction is rather delicate since in concurrent systems there are many different interactions that can happen. For example, we need to schedule actions of process q, using $ch(a)$ actions, before the actions of process r. The reason is the following. First, we need to make all r-actions uncontrollable, so that the environment could choose any play respecting the chosen r-local strategy. Now, if we allowed controllable q-actions to be eligible at the same time as r-actions, then the control strategy for automaton $\mathcal{A}^\triangledown$ would be to propose nothing and force the environment to play the r-actions. This would allow the controller of $\mathcal{A}^\triangledown$ to force the advancement of the simulation of r and get information that is impossible to obtain by the controller of $\mathcal{A}$.

Together with Theorem 10, the theorem below implies our main Theorem 6.

▶ **Theorem 13.** *For every r-short Zielonka automaton $\mathcal{A}$ and every local, ω-regular correctness condition: there is a correct covering controller for $\mathcal{A}$ iff there is a correct covering controller for $\mathcal{A}^\triangledown$. The size of $\mathcal{A}_q^\triangledown$ is polynomial in the size of $\mathcal{A}_q$ and exponential in the size of $\mathcal{A}_r$.*

3.3 Proof of Theorem 13

Given the space limit we can only give a sketch of one direction of the proof of Theorem 13. We consider the right-to-left direction. Given a correct controller $\mathcal{D}^\triangledown$ for $\mathcal{A}^\triangledown$, we show how to construct a correct controller $\mathcal{D}$ for $\mathcal{A}$. By Lemma 8 we can assume that $\mathcal{D}^\triangledown$ is covering.

The components $\mathcal{D}_p$ for $p \neq q, r$ are the same as in $\mathcal{D}^\triangledown$. So it remains to define $\mathcal{D}_q$ and $\mathcal{D}_r$. The states of $\mathcal{D}_q$ and $\mathcal{D}_r$ are obtained from states of $\mathcal{D}_q^\triangledown$. We need only certain states of $\mathcal{D}_q^\triangledown$, namely those d_q whose projection $\pi^\triangledown(d_q)$ in $\mathcal{A}_q^\triangledown$ has four components, we call them *true states* of $\mathcal{D}_q^\triangledown$:

$$ts(\mathcal{D}_q^\triangledown) = \{d_q \in \mathcal{D}_q^\triangledown \mid \pi^\triangledown(d_q) \text{ is of the form } (s_q, a, s_r, f)\}.$$

Figure 4 presents an execution of $\mathcal{A}^\triangledown$ controlled by $\mathcal{D}^\triangledown$. We can see that d_2 is a true state, and d_3 is not.

The set of states of $\mathcal{D}_q$ is just $ts(\mathcal{D}_q^\triangledown)$, while the states of $\mathcal{D}_r$ are pairs (d_q, x) where d_q is a state from $ts(\mathcal{D}_q^\triangledown)$ and $x \in (\Sigma_r^{loc})^*$ is a sequence of local r-actions that is possible from d_q in $\mathcal{D}^\triangledown$, in symbols $d_q \xrightarrow{x}$. We will argue later that such sequences are uniformly bounded. The initial state of $\mathcal{D}_q$ is the state d_q^1 reached from the initial state of $\mathcal{D}_q^\triangledown$ by the (unique) transitions of the form $ch(f_0), ch(a_0)$. The initial state of $\mathcal{D}_r$ is (d_q^1, ε). The local transitions for $\mathcal{D}_r$ are $(d_q, x) \xrightarrow{b} (d_q, xb)$, for every $b \in \Sigma_r^{loc}$ and $d_q \xrightarrow{xb}$.

Before defining the transitions of $\mathcal{D}_q$ let us observe that if $d_q \in \mathcal{D}_q^\triangledown$ is not in $ts(\mathcal{D}_q^\triangledown)$ then only one controllable transition is possible from it. Indeed, as $\mathcal{D}^\triangledown$ is a covering controller, if $\pi^\triangledown(d_q)$ is of the form (s_q, s_r) then there can be only an outgoing transition on a letter of the form $ch(f)$. Similarly, if $\pi^\triangledown(d_q)$ is of the form (s_q, s_r, f) then only a $ch(a)$ transition is possible. Since both $ch(f)$ and $ch(a)$ are controllable, we can assume that in $\mathcal{D}_q^\triangledown$ there is no state with two outgoing transitions on a letter of this form. For a state $d_q \in \mathcal{D}_q^\triangledown$ not in $ts(\mathcal{D}_q^\triangledown)$ we will denote by $ts(d_q)$ the unique state of $ts(\mathcal{D}_q^\triangledown)$ reachable from d_q by one or two transitions of the kind $\xrightarrow{ch(f)}$ or $\xrightarrow{ch(a)}$, depending on the cases discussed above. For example, going back to Figure 4, we have $ts(d_3) = d_4$.

We now describe the q-actions possible in $\mathcal{D}$.

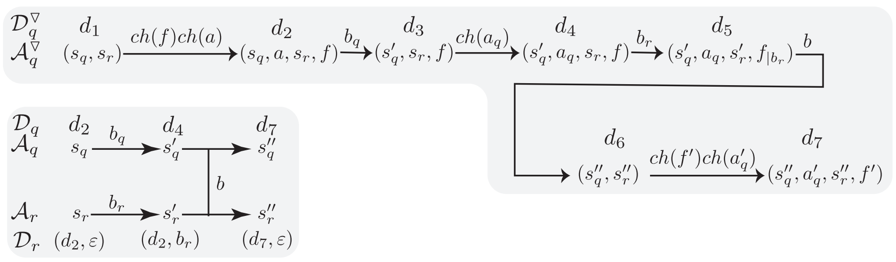


Figure 4 Decomposing controller $\mathcal{D}_q^\nabla$ into $\mathcal{D}_q$ and $\mathcal{D}_r$.

- Local q-action $b \in \Sigma_q^{loc}$: $d_q \xrightarrow{b} ts(d'_q)$ if $d_q \xrightarrow{b} d'_q$ in $\mathcal{D}_q^\nabla$.
- Communication $b \in \Sigma_q \cap \Sigma_p$ between q and $p \neq r$: $(d_p, d_q) \xrightarrow{b} (d'_p, ts(d'_q))$ if $(d_p, d_q) \xrightarrow{b} (d'_p, d'_q)$ in $\mathcal{D}^\nabla$.
- Communication $b \in \Sigma_q \cap \Sigma_r$ of q and r: $(d_q^1, (d_q^2, x)) \xrightarrow{b} (ts(d''_q), (ts(d''_q), \varepsilon))$ if $d_q^1 \xrightarrow{x} d'_q \xrightarrow{b} d''_q$ in $\mathcal{D}_q^\nabla$; observe that $\xrightarrow{x}$ is a sequence of transitions.

In Figure 4 transitions on b_1 and b are examples of transitions of the first and the third type, respectively. In the last item the transition does not depend on d_q^2 since, informally, d_q^1 has been reached from d_q^2 by a sequence of actions independent of r. The condition $d_q^1 \xrightarrow{x} d'_q \xrightarrow{b} d''_q$ simulates the order of actions where all local r-actions come after the other actions of q, then we add a communication between q and r.

The next lemma says that $\mathcal{D}$ is a covering controller for $\mathcal{A}$. Since $\mathcal{A}$ is assumed to be r-short, the lemma also gives a bound on the length of sequences in the states of $\mathcal{D}_r$.

▶ **Lemma 14.** *If $\mathcal{D}^\nabla$ is a covering controller for $\mathcal{A}^\nabla$ then $\mathcal{D}$ is a covering controller for $\mathcal{A}$.*

As $\mathcal{D}$ is covering, to prove that $\mathcal{D}$ is correct we need to show that all its maximal runs satisfy the correctness condition. For this we will construct for every run of $\mathcal{D}$ a corresponding run of $\mathcal{D}^\nabla$. The following definition and lemma tells us that it is enough to look at the runs of $\mathcal{D}$ of a special form.

▶ **Definition 15** (*slow*)**.** We define $slow_r(\mathcal{D})$ as the set of all sequences labeling runs of $\mathcal{D}$ of the form $y_0 x_0 a_1 \cdots a_k y_k x_k a_{k+1} \ldots$ or $y_0 x_0 a_1 \cdots y_{k-1} x_{k-1} a_k x_k y_\omega$, where $a_i \in \Sigma_q \cap \Sigma_r$, $x_i \in (\Sigma_r^{loc})^*$, $y_i \in (\Sigma \setminus \Sigma_r)^*$, and $y_\omega \in (\Sigma \setminus \Sigma_r)^\omega$

▶ **Lemma 16.** *A covering controller $\mathcal{D}$ is correct for $\mathcal{A}$ iff for all $w \in slow_r(\mathcal{D})$, $run(w)$ satisfies the correctness condition inherited from $\mathcal{A}$.*

For every sequence $w \in slow_r(\mathcal{D})$ as in Definition 15 we define the sequence $\chi(w) \in (\Sigma^\nabla)^\infty$, and show that it is a run of $\mathcal{D}^\nabla$. The definition is by induction on the length of w. Let $\chi(\varepsilon) = ch(f_0)\, ch(a_0)$, where f_0 and a_0 are determined by the initial q-state of $\mathcal{D}^\nabla$.

$$\text{For } w \in \Sigma^*, b \in \Sigma \text{ let} \qquad \chi(wb) = \begin{cases} \chi(w)b & \text{if } b \notin \Sigma_q \\ \chi(w)b\, ch(a) & \text{if } b \in \Sigma_q \setminus \Sigma_r \\ \chi(w)b\, ch(f)\, ch(a) & \text{if } b \in \Sigma_q \cap \Sigma_r. \end{cases}$$

where a and f are determined by the state reached by $\mathcal{D}^\nabla$ on $\chi(w)b$. The next lemma implies the correctness of the construction, and at the same time confirms that the above definition makes sense, that is, the needed runs of $\mathcal{D}^\nabla$ are defined.

▶ **Lemma 17.** *For every sequence $w \in slow_r(\mathcal{D})$ we have that $\mathcal{D}^\triangledown$ has a run on $\chi(w)$. This run is maximal in $\mathcal{D}^\triangledown$ if $run(w)$ is maximal in $\mathcal{D}$. In consequence, if $\mathcal{D}^\triangledown$ is a correct covering controller for $\mathcal{A}^\triangledown$, then $\mathcal{D}$ is a correct covering controller for $\mathcal{A}$.*

4 Conclusion

We have considered a model obtained by instantiating Zielonka automata into the supervisory control framework of Ramadge and Wonham [17]. The result is a distributed synthesis framework that is both expressive and decidable in interesting cases. To substantiate we have sketched how to encode threaded boolean programs with compare-and-swap instructions into our model. Our main decidability result shows that the synthesis problem is decidable for hierarchical architectures and for all local omega-regular specifications. Recall that in the Pnueli and Rosner setting essentially only pipeline architectures are decidable, with an additional restriction that only the first and the last process in the pipeline can handle environment inputs. In our case all processes can interact with the environment.

The synthesis procedure presented here is in k-EXPTIME for architectures of depth k, in particular it is EXPTIME for the case of a one server communicating with clients who do not communicate between each other. From [7] we know that these bounds are tight.

This paper essentially closes the case of tree architectures introduced in [7]. The long standing open question is the decidability of the synthesis problem for all architectures [5].

References

1 A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

2 Julian Bradfield and Colin Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *The Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.

3 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. In *Proc. of GandALF*, EPTCS, pages 217–230, 2014.

4 Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *LICS*, pages 321–330, 2005.

5 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS*, volume 3328 of *LNCS*, pages 275–286, 2004.

6 Paul Gastin and Nathalie Sznajder. Fair synthesis for asynchronous distributed systems. *ACM Transactions on Computational Logic*, 2013.

7 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In *Proceedings of ICALP'13*, 2013.

8 Susanne Graf, Doron Peled, and Sophie Quinton. Achieving distributed control through model checking. *Formal Methods in System Design*, 40(2):263–281, 2012.

9 Julian Gutierrez and Glynn Winskel. Borel determinacy of concurrent games. In *Proceedings of CONCUR'13*, 2013.

10 O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *LICS*, 2001.

11 P. Madhusudan and P. S. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP'01*, volume 2076 of *LNCS*, pages 396–407. Springer, 2001.

12 P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *FSTTCS*, volume 3821 of *LNCS*, pages 201–212, 2005.

13 Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. *TCS*, 358(2-3):200–228, 2006.

14 Madhavan Mukund and Milind A. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. *Distributed Computing*, 10(3):137–148, 1997.

15 Anca Muscholl and Sven Schewe. Unlimited decidability of distributed synthesis with limited missing knowledge. In *Proceedings of MFCS'13*, 2013.

16 A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *31th IEEE Symposium Foundations of Computer Science (FOCS 1990)*, pages 746–757, 1990.

17 P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(2):81–98, 1989.

18 Igor Walukiewicz. Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, 2001.

19 W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.

Verification of Dynamic Register Automata[*]

Parosh Aziz Abdulla[1], Mohamed Faouzi Atig[1], Ahmet Kara[2], and Othmane Rezine[1]

1 **Uppsala University, Sweden**
`{parosh,mohamed_faouzi.atig,othmane.rezine}@it.uu.se`

2 **TU Dortmund University, Germany**
`ahmet.kara@cs.tu-dortmund.de`

Abstract

We consider the verification problem for Dynamic Register Automata (Dra). Dra extend classical register automata by process creation. In this setting, each process is equipped with a finite set of registers in which the process IDs of other processes can be stored. A process can communicate with processes whose IDs are stored in its registers and can send them the content of its registers. The state reachability problem asks whether a Dra reaches a configuration where at least one process is in an error state. We first show that this problem is in general undecidable. This result holds even when we restrict the analysis to configurations where the maximal length of the simple paths in their underlying (un)directed communication graphs are bounded by some constant. Then we introduce the model of *degenerative* Dra which allows non-deterministic reset of the registers. We prove that for every given Dra, its corresponding degenerative one has the same set of reachable states. While the state reachability of a degenerative Dra remains undecidable, we show that the problem becomes decidable with nonprimitive recursive complexity when we restrict the analysis to strongly bounded configurations, i. e. configurations whose underlying undirected graphs have bounded simple paths. Finally, we consider the class of strongly safe Dra, where all the reachable configurations are assumed to be strongly bounded. We show that for strongly safe Dra, the state reachability problem becomes decidable.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Verification, Reachability problem, Register automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.653

1 Introduction

Register automata are a well-known computational model for languages over infinite alphabets (e.g. [20, 23, 24]). A register automaton is a finite state automaton equipped with a finite set of registers which can store data for later comparison. The expressive power and algorithmic properties of this model are well-studied (see e.g., [6, 23, 24, 28]). In addition, several works consider the relationship between different classes of register automata and logics for data words and trees (see e.g., [14, 15, 21, 19]).

Recently, register automata have been extended with dynamic creation of processes [8, 7]. In this setting, the behaviour of each process is described by a register automaton. Each process has a unique identifier (ID). The registers of each process are used to store the IDs of other process. The IDs stored in the registers of a process p correspond to the processes

[*] Supported by the Uppsala Programming for Multicore Architectures Research Center (UPMARC) and the Programming Platform for Future Wireless Sensor Networks Project (PROFUN). The third author acknowledges the financial support by the German DFG under grant SCHW 678/4-2.

known by p. Each process can perform two types of actions: (i) creating a new process and (ii) exchanging messages and IDs with other processes. The class of extended register automata can be used as: (1) a model of programs with process creation where the network topology and the number of involved processes are not known in advance but change dynamically [8], and (2) an implementation model for Dynamic Message Sequence Charts [8, 7].

In this paper, we consider the verification problem for Dynamic Register Automata (DRA), where the communication between processes is *synchronous* (i. e., *rendezvous* based)[1]. The synchronous communication involves two processes: *sender* and *receiver*. Besides creating new processes, each process can send a message from a finite alphabet or an ID from one of its registers (or its own ID). The receiver process can synchronize over the sent message or store the incoming ID in its own registers. Thus, the system may create an unbounded number of processes, and the communication topology can change dynamically.

As argued in [10, 11], the *state reachability problem* or the *coverability problem* are adequate for capturing several interesting properties that arise in communicating systems (e.g., Ad-Hoc networks). The problem consists in checking whether the system can start from a given initial configuration and evolve to reach a configuration in which at least one of the processes is in a given error state. To the best of our knowledge, this is the first work addressing the control state reachability problem for a class of dynamic register automata.

In this paper, we first show that the state reachability problem is undecidable even in the case where each process is equipped with only one register. Then, an important task is to identify subclasses of DRA for which algorithmic verification is possible. Inspired by some recent works on the verification of Ad-Hoc networks [10, 1], we consider a restricted version of the verification problem where we restrict the analysis to only *bounded* configurations, in which the maximum length of directed simple paths in the induced communication graph is bounded by a given natural number k. The communication graph represents the connectivity of the network induced by a DRA. In this graph each process is represented by a node and there is an edge from a node u to a node v if the process corresponding to u knows the process corresponding to v. It turns out that the verification problem remains undecidable for bounded DRA with at least two registers. Moreover, this undecidability holds even if we restrict the analysis to *strongly bounded* configurations, in which we require that the maximum length of simple paths in the *undirected* communication graph (i. e., regardless of the direction of the edges) is bounded (unlike the case of Ad-hoc networks [10, 11, 13, 1]).

Then, we introduce the model of *degenerative* DRA, a DRA in which any register can be reset in non-deterministic way. Degenerative DRA can be used to model unexpected loss of communication links in mobile Ad-hoc networks. Given a DRA, we associate a degenerative counterpart by allowing reset transitions at every state and for every register of the DRA. We show that the degenerative counterpart of a DRA represents an over-approximation of the original DRA in terms of reachable states. We prove that the approximation is exact by showing that the degenerative DRA does not expose more states than its non-degenerative counterpart. This implies that the reachability problem for degenerative DRA is also undecidable. Therefore, we consider the subclass of strongly bounded degenerative DRA. We show that degenerative DRA is a (*strict*) over-approximation of its non-degenerative counterpart (in terms of reachable states) when both are restricted to strongly bounded communication graphs. We also show that the state reachability problem for the class of strongly bounded degenerative DRA is decidable. The decidability proof is carried out by defining a symbolic backward reachability analysis based on a non-trivial instantiation of the

[1] Observe that in [7, 8], processes of DRA communicate asynchronously via (bounded) FIFO channels.

framework of well structured transition systems [2, 16]. Furthermore, we show that state reachability for the class of strongly bounded degenerative DRA is nonprimitive recursive by a reduction from reachability for lossy counter machines [25]. Hence, the class of strongly bounded degenerative DRA represents a good candidate for a decidable subclass of DRA.

We point out that bounded DRA with only one register is in fact strongly bounded. Thus, the state reachability problem for bounded degenerative 1-register-DRA is also decidable.

Finally, we introduce (strongly) safe DRA where we assume that all the reachable configurations are (strongly) bounded. We show that the state reachability problem for strongly safe DRA becomes decidable while the undecidability still holds for safe DRA.

Related work. Communicating finite state machines [9] are a well-known computational model for distributed systems where processes communicate through unbounded channels. They serve, for instance, as an implementation model for Message Sequence Charts with finitely many processes [5, 4, 18]. Several works address the verification problem, in particular the state reachability problem, of different classes of this model [3, 22, 17]. However, in contrary to our model, in most of these settings a *fixed* number of processes is considered which restricts their applicability for dynamic systems.

Communicating finite state machines are also used as a formal model for wireless Ad-Hoc networks [26, 27, 10, 11, 13, 1]. Every process in an Ad-Hoc network can perform local, (selective) broadcast and receive actions. While processes in a DRA perform 1-to-1 communications, broadcast actions in Ad-Hoc networks involve multiple processes. By performing a broadcast action a process sends a message to all its neighbour processes (whose number is not bounded *a priori*). An important question in the realm of Ad-Hoc networks is the state reachability problem, parametrized by the number of involved processes and by the network topology: is there a number of processes and a network topology such that after a finite number of transitions one process reaches a special state? Even though [26] and [10] consider models where the topology of the network can change, the processes cannot perform process creation, thus, the number of interacting processes is (arbitrary but) fixed.

Broadcast networks of register automata are introduced in [12]. The model is similar to DRA in the sense that the automata are equipped with a finite set of registers which can store some data. Besides this fact, the model of [12] does not support process creation and exchanging process ID does not affect the network topology.

2 Preliminaries

Let $\mathbb{N}$ denote the set of natural numbers. Let A and B be two sets. We use $|A|$ to denote the cardinality of A ($|A| = \omega$ if A is infinite). For a partial function $g : A \rightharpoonup B$ and $a \in A$, we write $g\,(a) = \bot$ if g is undefined on a. We use $\bot_A$ to denote the partial function which is undefined on all elements of A, i. e. $\bot_A\,(a) = \bot$ for every $a \in A$. Given a (partial) function $f : A \rightharpoonup B$, $a \in A$ and $b \in B$, we denote by $f[a \leftarrow b]$ the function f' defined by $f'\,(a) = b$ and $f'\,(a') = f\,(a')$ for all $a' \in A$ with $a' \neq a$.

A *transition system* $\mathcal{T}$ is a triple $\langle C, C_{init}, \longrightarrow\rangle$, where C is a set of *configurations*, $C_{init} \subseteq C$ is an *initial* set of configurations, and $\longrightarrow \subseteq C \times C$ is a *transition relation*. We write $c_1 \longrightarrow c_2$ when $\langle c_1, c_2\rangle \in \longrightarrow$ and $\longrightarrow^*$ to denote the reflexive transitive closure of $\longrightarrow$. For every $i \in \mathbb{N}$, we use $\longrightarrow^i$ to denote the i-times composition of $\longrightarrow$. A configuration $c \in C$ is said *reachable* in $\mathcal{T}$ if there is $c_{init} \in C_{init}$ such that $c_{init} \longrightarrow^* c$.

A *directed labeled graph* (or simply *graph*) G is a tuple $\langle V, \Sigma_v, \Sigma_e, \lambda, E\rangle$ where V is a finite set of vertices, Σ_v is a set of vertex labels, Σ_e is a set of edge labels, $\lambda : V \to \Sigma_v$ is the vertex labeling function, and $E \subseteq V \times \Sigma_e \times V$ is the set of edges. A *path* in G is a finite sequence of vertices $\pi = v_1 v_2 \ldots v_k, k \geq 1$, where, for every $i : 1 \leq i < k$, there is an $a \in \Sigma_e$ such that $\langle v_i, a, v_{i+1} \rangle \in E$. We say that π is *simple* if all vertices in π are different, i. e. $v_i \neq v_j$ for all $i, j : 1 \leq i < j \leq k$, and we define $\texttt{length}\,(\pi) := k - 1$. We define the diameter of G, denoted by $\oslash(G)$, to be the largest k such that there is a simple path π in G with $\texttt{length}\,(\pi) = k$.

3 Dynamic Register Automata

A *Dynamic Register Automaton* Dra consists of a set of processes that exchange messages and create new processes. Each process is modelled as a finite state automaton equipped with a finite set of registers. A register may contain the identifier (ID) of another process. A process can perform a *local* action that changes its current state. It can also create (or spawn) a new process, allowing the number of processes to increase over time. Communication is allowed between two processes given that the sender has the ID of the receiver in one of its registers. A process can send a message from a finite alphabet, its own ID as well as the content of one of its registers. Below, we describe the syntax of Dra and introduce the subclass of *degenerative* Dra where any register can be reset in a non-deterministic way. Then, we define the operational semantics of a Dra, and its state reachability problem.

Definition. A Dra D is a tuple $\langle Q, q_0, M, X, \delta \rangle$ where Q is a finite set of control states, $q_0 \in Q$ is the initial state, M is a finite set of messages, $X = \{x_1, \ldots, x_n\}$ is a finite set of registers, and δ is a set of transitions, each of the form $\langle q_1, \texttt{action}, q_2 \rangle$ where $q_1, q_2 \in Q$ are control states and $\texttt{action}$ is of one of the following forms: (i) τ (local action), (ii) $x \hookleftarrow \texttt{create}(q, y)$ where $x, y \in X$ and $q \in Q$, creates a new process with a fresh ID in state q, stores the ID of the new process in register x of the creator process, and stores the ID of the creating process in register y of the new process, (iii) $x!\,\langle m \rangle$ where $x \in X, m \in M$, sends message m to the process whose ID is stored in register x, (iv) $x!\,\langle y \rangle$ where $x \in X, y \in X \cup \{\texttt{self}\}$, sends either the ID contained in register y or the ID of the process itself ($\texttt{self}$) to the process whose ID is stored in x, (v) $x?\,\langle m \rangle$ where $x \in X, m \in M$ (selective message reception), receives message m from the process whose ID is stored in register x, (vi) $\star?\,\langle m \rangle$ where $m \in M$ (nonselective message reception), receives message m from some other process, (vii) $x?\,\langle y \rangle$ where $x \in X, y \in X$ (selective ID reception), receives an ID to be stored in register y from a process whose ID is stored in x, (viii) $\star?\,\langle y \rangle$ where $y \in X$ (nonselective ID reception), receives an ID to be stored in register y from some other process, and (ix) $\texttt{reset}\,\langle x \rangle$ where $x \in X$, resets register x so that it becomes undefined.

The Dra D is *degenerative* if for every state $q \in Q$ and register $x \in X$, $\langle q, \texttt{reset}\,\langle x \rangle, q \rangle \in \delta$. Given a Dra $D = \langle Q, q_0, M, X, \delta \rangle$, we define its degenerative counterpart Dra $\texttt{Deg}\,(D)$ by the tuple $\langle Q, q_0, M, X, \delta' \rangle$ with $\delta' = \delta \cup \{\langle q, \texttt{reset}\,\langle x \rangle, q \rangle \mid q \in Q, x \in X\}$.

Configuration. We use $\mathcal{P}$ to denote the domain of all possible process IDs. Let $D = \langle Q, q_0, M, X, \delta \rangle$ be a Dra. We define a configuration c as the tuple $\langle \texttt{procs}, \texttt{s}, \texttt{r} \rangle$, where $\texttt{procs} \subseteq \mathcal{P}$ is a finite set of processes, $\texttt{s} : \mathcal{P} \rightharpoonup Q$ maps each process $p \in \texttt{procs}$ to its current state and $\texttt{r} : \mathcal{P} \rightharpoonup \{X \rightharpoonup \texttt{procs}\}$ is a partial function that maps every process $p \in \texttt{procs}$ to its registers contents. For two processes $p_1, p_2 \in \texttt{procs}$ and $x \in X$, $\texttt{r}\,(p_1)\,(x) = p_2$ means that register x of p_1 contains the ID of p_2. If $\texttt{r}\,(p_1)\,(x)$ is not defined then register x of p_1 is empty. We use $q \in c$ to denote that there exists a process $p \in \texttt{procs}$ such that $\texttt{s}\,(p) = q$. The set of all possible configurations of D is denoted by $C(D)$. A configuration $c = \langle \texttt{procs}, \texttt{s}, \texttt{r} \rangle \in C(D)$ is said to be *initial* if it contains exactly one process (i. e., $\texttt{procs} = \{p\}$ for some $p \in \mathcal{P}$), which is in the initial state ($\mathtt{s}(p) = q_0$) and whose registers are empty ($\mathtt{r}(p) = \bot_X$). The set of initial configurations is denoted by $C_{init}(D)$.

Encoding of Configurations. The encoding of a configuration c is a graph $\mathtt{enc}(c)$ that models its register mappings. Every process in the encoding is represented by a vertex labeled with the state of the process. Furthermore, there is an edge from vertex u to vertex v labeled with $x \in X$ if the process corresponding to u has the ID of the process corresponding to v in its register x. Formally, the encoding of a configuration $c = \langle \mathtt{procs}, \mathtt{s}, \mathtt{r} \rangle$ is defined as the graph $\mathtt{enc}(c) := \langle \mathtt{procs}, Q, X, \mathtt{s}, E = \{\langle p, x, p' \rangle | \, \mathtt{r}(p)(x) = p'\} \rangle$.

Transition Relation. We define a transition relation $\rightarrow_D$ on the set $C(D)$ of configurations of the DRA D. Given two configurations $c = \langle \mathtt{procs}, \mathtt{s}, \mathtt{r} \rangle, c' = \langle \mathtt{procs}', \mathtt{s}', \mathtt{r}' \rangle \in C(D)$, we have $c \rightarrow_D c'$ if one of the following conditions holds:

`Local` There is a transition $\langle q_1, \tau, q_2 \rangle \in \delta$ and a process $p \in \mathtt{procs}$ such that (i) $\mathtt{procs}' = \mathtt{procs}$ and $\mathtt{r}' = \mathtt{r}$, i. e., the processes and registers are left unchanged, (ii) $\mathtt{s}(p) = q_1$, and (iii) $\mathtt{s}' = \mathtt{s}[p \leftarrow q_2]$. A local transition changes the state of (at most) one process.

`Create` There is a transition $\langle q_1, x \longleftarrow \mathtt{create}\langle q, y \rangle, q_2 \rangle \in \delta$ and a process $p \in \mathtt{procs}$ such that (i) $\mathtt{s}(p) = q_1$, i. e., p is in state q_1, (ii) $\mathtt{procs}' = \mathtt{procs} \cup \{p'\}$ for some process $p' \notin \mathtt{procs}$, i. e., a new process p' is created, (iii) $\mathtt{s}' = \mathtt{s}[p \leftarrow q_2][p' \leftarrow q]$, i. e., process p' is spawned in state q, while the new state of process p is q_2, and (iv) $\mathtt{r}' = \mathtt{r}[p \leftarrow \mathtt{r}(p)[x \leftarrow p']][p' \leftarrow \bot_X[y \leftarrow p]]$, i. e., register x of process p is assigned the ID of the new process p' and register y of process p' is assigned the ID of process p.

`Selective message sending` There are two different processes $p, p' \in \mathtt{procs}$ and two transitions $\langle q_1, x!\langle m \rangle, q_2 \rangle, \langle q_3, y?\langle m \rangle, q_4 \rangle \in \delta$ such that (i) $\mathtt{s}(p) = q_1$ and $\mathtt{s}(p') = q_3$, i. e., p and p' are in states q_1 and q_3, respectively, (ii) $\mathtt{r}(p)(x) = p'$ and $\mathtt{r}(p')(y) = p$, i. e., the sender p has the ID of p' in its register x and the receiver p' has the ID of p in its register y, (iii) $\mathtt{s}' = \mathtt{s}[p \leftarrow q_2][p' \leftarrow q_4]$, i. e., the states of both processes p and p' are updated simultaneously, and (iv) $\mathtt{r}' = \mathtt{r}$, i. e., the registers are unchanged.

`Selective ID sending` There are two different processes $p, p' \in \mathtt{procs}$ and two transitions $\langle q_1, x!\langle z_1 \rangle, q_2 \rangle, \langle q_3, y?\langle z_2 \rangle, q_4 \rangle \in \delta$ such that (i) $\mathtt{s}(p) = q_1$ and $\mathtt{s}(p') = q_3$, (ii) $\mathtt{r}(p)(x) = p'$ and $\mathtt{r}(p')(y) = p$, (iii) $\mathtt{s}' = \mathtt{s}[p \leftarrow q_2][p' \leftarrow q_4]$, (iv) either $z_1 = \mathtt{self}$ or there exist $p'' \in \mathtt{procs}$ such that $\mathtt{r}(p)(z_1) = p''$, i. e., the ID to be sent should be the ID of some process, and (v) $\mathtt{r}' = \mathtt{r}[p' \leftarrow \mathtt{r}(p')[z_2 \leftarrow p]]$ if $z_1 = \mathtt{self}$ or $\mathtt{r}' = \mathtt{r}[p' \leftarrow \mathtt{r}(p')[z_2 \leftarrow p'']]$ otherwise, i. e., register z_2 of p' is updated with what it receives from p.

`Register resetting` There is a transition $\langle q_1, \mathtt{reset}\langle x \rangle, q_2 \rangle \in \delta$ and a process $p \in \mathtt{procs}$ such that (i) $\mathtt{s}(p) = q_1$ and $\mathtt{s}' = \mathtt{s}[p \leftarrow q_2]$, and (ii) $\mathtt{r}' = \mathtt{r}[p \leftarrow \mathtt{r}(p)[x \leftarrow \bot]]$, i. e., register x of process p' is reset.

The only difference between `Nonselective message sending` and `Nonselective ID sending` and their selective counterparts is that the receiver does not need to know the sender, i. e., the ID of the sending process does not have to be in the registers of the receiver.

For a DRA $D = \langle Q, q_0, M, X, \delta \rangle$, we use $\xrightarrow{\mathtt{reset}}_D \subseteq C(D) \times C(D)$ to denote the set of transitions induced by the set of `Register resetting` transitions in δ of the form $\langle q, \mathtt{reset}\langle x \rangle, q \rangle$ with $q \in Q$ and $x \in X$.

State Reachability. Let $\mathcal{T}(D)$ denote the transition system defined by the triple $\langle C(D), C_{init}(D), \longrightarrow_D \rangle$. Let $\mathtt{target} \in Q$ be a state of D. The state $\mathtt{target}$ is said to be reachable if there exists a reachable configuration that has a process in state $\mathtt{target}$. The state

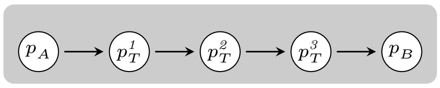


Figure 1 Transduction chain.

reachability problem consists in checking whether the state `target` is reachable or not. We use `StateReach`$(D,$ `target`$)$ to denote the state reachability problem for D and `target`.

It is obvious that any degenerative DRA is an over-approximation of its non-degenerative counterparts in terms of reachable states. Lemma 1 states that this approximation is exact.

▶ **Lemma 1.** *Let D be a* DRA*. Then, D and* `Deg`(D) *reach the same set of control states.*

4 State Reachability for (Degenerative) Dra

In the following, we show that the state reachability for (degenerative) DRA with at least one register is undecidable.

▶ **Theorem 2.** *Given a (degenerative)* DRA *$D = \langle Q, q_0, M, X, \delta\rangle$ and a state $q_f \in Q$,* `StateReach`(D, q_f) *is undecidable. This undecidability holds even in the case where $|X| = 1$.*

The proof proceeds by reduction from the `Transd` problem defined below.

The `Transd` Problem. A *transducer* T is a tuple $\langle Q, q_{init}, \Sigma, \delta, F\rangle$ where Q is a finite set of *states*, q_{init} is the *initial* state, Σ is a finite *alphabet*, $\delta \subseteq Q \times \Sigma \times \Sigma \times Q$ is the *transducer transition relation*, and F is the set of *accepting* states. Every transition $t \in \delta$ gets as input some symbol $a \in \Sigma$ and outputs another symbol $b \in \Sigma$. The transducer transition relation δ induces on Σ^* a binary relation *Rel*, where $w\, Rel\, w'$ if w' is the output of T when accepting w. Given a word $w \in \Sigma^*$, let $T(w) := \{v \in \Sigma^* |\, w\, Rel\, v\}$ denote the set of any possible transduction of w by T. We extend the notion of transduction to a language $L \subseteq \Sigma^*$ by defining $T(L) := \bigcup_{w\in L} T(w)$. In an iterative way, we define for $i \in \mathbb{N}$ the i^{th} transduction of L as $T^0(L) := L$ and $T^{i+1}(L) := T(T^i(L))$. Given a finite state automaton A over the alphabet Σ, we denote by $L(A)$ the regular language accepted by A. An instance of the problem `Transd` consists of two finite state automata A and B, and a transducer T, all over the same alphabet Σ. In `Transd` it is checked whether there is a natural number $i \in \mathbb{N}$ such that $T^i(L(A)) \cap L(B) \neq \emptyset$. The problem `Transd` is known to be undecidable [1].

A Sketched proof of Thm. 2. Given an instance of `Transd`, i. e. two automata A and B and a transducer T over the same alphabet, the encoding of `Transd` into the state reachability problem of DRA consists of constructing a *transduction chain*, where the first element of the chain is a process p_A encoding A, the last one is a process p_B encoding B and all intermediate elements are processes p_T^i encoding T (Figure 1). The simulation of the transduction works as follows: The first process p_A sends a word $w \in \Sigma^*$ symbol by symbol to its successor in the chain. If w is a word accepted by A, p_A sends a special acceptance symbol to its successor. Meanwhile, each intermediate process simulating T sends for every incoming symbol from Σ a corresponding output symbol to its successor. If it gets the acceptance symbol it checks whether the so far received word is accepted by T. If it is the case, it transmits the acceptance symbol to the next process. At the reception of the acceptance symbol, the last process p_B in the chain checks whether the received word is accepted by B. If it is the case, it moves to the state q_f, if not, it moves to an error (deadlock) state. Note that if there are no intermediate processes simulating T, process p_A sends the symbols directly to p_B. It can be shown by induction that there exists an $i \geq 0$ with $T^i(L(A)) \cap L(B) \neq \emptyset$ if and only if a transduction chain of length $i+2$ which reaches q_f can be constructed. Note that processes in the chain do not need more than one register and that a correct transduction chain can also be constructed by a degenerative Dra.

5 Bounded (Degenerative) Dra

The reduction from `Transd` to the state reachability for (degenerative) Dra relies on the fact that the transduction chain can be made as long as desired, allowing for $i \in \mathbb{N}$ in $T^i(L(A))$ to be as large as needed. One way to break the transducer chain proof would be to bound the diameter of the configuration encodings. In the following we show that this condition is still not sufficient. Let us first define a transition system where only configurations with bounded diameter are allowed. Let k be a natural number, D a Dra and $\mathcal{T}(D) = \langle C(D), C_{init}(D), \longrightarrow_D \rangle$ its corresponding transition system. We say that a configuration $c \in C(D)$ is k-bounded if the diameter of its encoding is bounded by k, i.e $\oslash(\texttt{enc}(c)) \leq k$. Given a set $B \subseteq C(D)$ of configurations, we use $(B\Box k)$ to denote the set of k-bounded configurations in B. The restriction of $\longrightarrow_D$ to the set $C(D)\Box k$ of k-bounded configurations is denoted by $\longrightarrow_D^{\Box k} := \longrightarrow_D \cap((C(D)\Box k)\times(C(D)\Box k))$. We use $\mathcal{T}^{\Box k}(D)$ to denote the resulting transition system defined by $\langle (C(D)\Box k), (C_{init}(D)\Box k), \longrightarrow_D^{\Box k} \rangle$. Given a state $\texttt{target} \in Q$, the *k-bounded state reachability problem* consists in checking whether a configuration c with $\texttt{target} \in c$ is reachable in $\mathcal{T}^{\Box k}(D)$. We use $\texttt{BoundedStateReach}(D, \texttt{target}, k)$ to denote the k-bounded state reachability problem. We prove the following result:

▶ **Theorem 3.** *Given a natural number $k \in \mathbb{N}$, a (degenerative)* Dra *$D = \langle Q, q_0, M, X, \delta \rangle$ and a state $q_f \in Q$, $\texttt{BoundedStateReach}(D, q_f, k)$ is undecidable. This undecidability still holds even if $k = 2$ and $|X| = 2$.*

The proof can be done by a reduction from the `Transd` problem. Observe that there is no straightforward reduction from Thm. 3 to Thm. 2 and vice-versa.

6 Strongly Bounded (Degenerative) Dra

As we have seen, bounding the diameter of the configuration encoding is insufficient to get decidability of the state reachability problem. Therefore, we consider a new constraint on the graph encoding of the configurations. The new constraint consists in restricting the set of configurations such that the diameter of their graph encodings is bounded by some natural number k, this time regardless of the direction of the edges in the graph. In order to formally specify the new constraint, let us introduce the class of label-free undirected graphs.

Label-free Undirected Graph. A *label-free undirected graph* G is a graph whose edges have no labels and no direction, i. e. G is a tuple $\langle V, \Sigma_v, \lambda, E \rangle$ where V is a finite set of vertices, Σ_v is a finite set of vertex labels, $\lambda : V \to \Sigma_v$ is a vertex labeling function and $E \subseteq \{\{u, v\} | \; u, v \in V\}$ is a set of unlabeled and undirected edges. Notions of simple path and diameter of a graph are extended in the natural way to label-free undirected graphs. Given a (directed) graph $G = \langle V, \Sigma_v, \Sigma_e, \lambda, E \rangle$, we use $\texttt{closure}(G) := \langle V, \Sigma_v, \lambda, F \rangle$ to denote the undirected graph obtained from G by removing directions and labels from its edges, i. e. $F := \{\{u, v\} | \; \langle u, a, v \rangle \in E\}$.

Strongly Bounded Configurations. Let k be a natural number, $D = \langle Q, q_0, M, X, \delta \rangle$ a Dra and $\mathcal{T}(D) = \langle C(D), C_{init}(D), \longrightarrow_D \rangle$ the transition system induced by D. Let c be a configuration in $C(D)$. We say that c is k-strongly bounded if $\oslash(\texttt{closure}(\texttt{enc}(c))) \leq k$. Given $B \subseteq C(D)$, we use $(B \Diamond k)$ to denote the set of k-strongly bounded configurations in B, i.e. $(B \Diamond k) := \{c \in B | \oslash(\texttt{closure}(\texttt{enc}(c))) \leq k\}$. We consider the transition relation $\longrightarrow_D^{\Diamond k}$ defined on $(C(D) \Diamond k)$ by $\longrightarrow_D^{\Diamond k} := \longrightarrow_D \cap((C(D) \Diamond k) \times (C(D) \Diamond k))$. We define the transition system $\mathcal{T}^{\Diamond k}(D) := \left\langle (C(D) \Diamond k), (C_{init}(D) \Diamond k), \longrightarrow_D^{\Diamond k} \right\rangle$. Given a state $\texttt{target} \in Q$, the *k-strongly bounded state reachability problem* consists in checking whether a configuration c with $\texttt{target} \in c$ is reachable in $\mathcal{T}^{\Diamond k}(D)$. We use $\texttt{StrongBoundStateReach}(D, \texttt{target}, k)$ to denote the k-strongly bounded state reachability problem.

▶ **Theorem 4.** *Given $k \in \mathbb{N}$, a Dra $D = \langle Q, q_0, M, X, \delta \rangle$ and a state $\texttt{target} \in Q$, $\texttt{StrongBoundStateReach}(D, \texttt{target}, k)$ is undecidable. This undecidability still holds even if $k = 4$ and $|X| = 2$.*

The proof of Thm. 4 can be established by a reduction from the reachability problem for Minsky's 2-counter machines.

▶ **Theorem 5.** *Given $k \in \mathbb{N}$, a degenerative Dra $D = \langle Q, q_0, M, X, \delta \rangle$ and a state $\texttt{target} \in Q$, $\texttt{StrongBoundStateReach}(D, \texttt{target}, k)$ is decidable and nonprimitive recursive.*

The decidability of the strongly bounded state reachability problem for degenerative Dra is established by a non-trivial instantiation of the framework of well-quasi-ordered systems [2, 16] (See Section 8). The nonprimitive recursive lower bound is carried out through a reduction from the reachability problem for *Lossy Counter Machines* [25].

Furthermore, it is clear that the set of k-strongly bounded reachable states by a Dra D is a (strict) subset of the set of k-strongly bounded reachable states by its degenerative Dra counterpart $\texttt{Deg}(D)$. Moreover, the set of k-strongly bounded reachable states by the degenerative Dra $\texttt{Deg}(D)$ is a subset of the set of reachable states by D. Thus, the strongly bounded reachability problem for $\texttt{Deg}(D)$ is a good under-approximation of the state reachability problem for D. This relation[2] between the strongly bounded reachability problems for a Dra D and its corresponding degenerative one $\texttt{Deg}(D)$ is given by the following observation:

▶ Observation 1. Let $k \in \mathbb{N}$ be a natural number, D a Dra, and $\texttt{target}$ a state of D. If $\texttt{target}$ is reachable in $\mathcal{T}^{\Diamond k}(D)$ then it is reachable in $\mathcal{T}^{\Diamond k}(\texttt{Deg}(D))$. Furthermore, if $\texttt{target}$ is reachable in $\mathcal{T}^{\Diamond k}(\texttt{Deg}(D))$ then there is $k' \geq k$ such that $\texttt{target}$ is reachable in $\mathcal{T}^{\Diamond k'}(D)$.

The decidability of bounded degenerative Dra with one register (see Corollary 7) can be inferred from Theorem 5 and the following lemma:

▶ **Lemma 6.** *Any k-bounded configuration of a Dra with one register is $2k$-strongly bounded.*

▶ **Corollary 7.** *Given a natural number $k \in \mathbb{N}$, a degenerative Dra $D = \langle Q, q_0, M, X, \delta \rangle$ with $|X| = 1$ and a state $\texttt{target} \in Q$, $\texttt{BoundedStateReach}(D, \texttt{target}, k)$ is decidable.*

7 (Strongly) Safe Dra

A k-strongly bounded Dra forbids transitions to configurations that are not k-strongly bounded. This allows to simulate zero tests of the Minsky's 2-counter machine in the proof

[2] Observe that this relation holds also for the (bounded) reachability problem.

of Thm. 4. Therefore, we introduce k-(strongly) safe DRA, with $k \in \mathbb{N}$, which is a DRA where we assume that all reachable configurations are k-(strongly) bounded. Formally, let D be a DRA and $\mathcal{T}(D)$ its induced transition system. The DRA D is said to be k-(strongly) safe iff every reachable configuration in $\mathcal{T}(D)$ is k-(strongly) bounded. We can state:

▶ Observation 2. If D is a k-strongly safe DRA then $\texttt{Deg}\,(D)$ is a k-strongly bounded DRA.

As an immediate consequence of Lemma 1, Observation 2 and Theorem 5, we infer:

▶ **Corollary 8.** *Given a k-strongly safe* DRA $D = \langle Q, q_0, M, X, \delta\rangle$ *and a state* $q_f \in Q$, $\texttt{StateReach}(D, q_f)$ *is decidable.*

However, the state reachability problem is still undecidable for k-safe (degenerative) DRA.

▶ **Theorem 9.** *Given a k-safe (degenerative)* DRA $D = \langle Q, q_0, M, X, \delta\rangle$ *and a state* $q_f \in Q$, $\texttt{StateReach}(D, q_f)$ *is undecidable.*

8 Strongly Bounded Degenerative Dra: Proof of Theorem 5

This section is devoted to the decidability proof of Theorem 5 by making use of the framework of *Well-Structured Transition Systems* (WSTS) [2, 16].

We briefly recall the framework of WSTS. Let C be a (possibly infinite) set and $\preccurlyeq$ be a *well-quasi order* on C. Recall that a well-quasi order on C is a binary relation over C that is reflexive and transitive and for every infinite sequence $(a_i)_{i\geq 0}$ of elements in C there exist $i, j \in \mathbb{N}$ such that $i < j$ and $a_i \preccurlyeq a_j$. A set $U \subseteq C$ is called *upward closed* if for every $a \in U$ and $b \in C$ with $a \preccurlyeq b$ we have $b \in U$. The upward closure of some set $U \subseteq C$ is defined as $U\!\uparrow := \{b \in C \mid \exists a \in U \text{ with } a \preccurlyeq b\}$. It is known that every upward closed set U can be characterised by a finite *minor set* $M \subseteq U$ such that (i) for every $a \in U$ there is $b \in M$ such that $b \preccurlyeq a$, and (ii) if $a, b \in M$ and $a \preccurlyeq b$ then $a = b$. We use $\texttt{min}$ to denote the function which for a given upward closed set U returns one minor set of U.

Let $\mathcal{T} = \langle C, C_{init}, \rightsquigarrow\rangle$ be a transition system and $\preccurlyeq$ be a well-quasi ordering on C. For a subset $U \subseteq C$ of configurations we define the set of predecessors of U as $\texttt{Pre}\,(U) := \{c \in C \mid \exists c_1 \in U, c \rightsquigarrow c_1\}$. For a configuration c we denote the set $\texttt{min}\,(\texttt{Pre}\,(\{c\}\!\uparrow) \cup \{c\}\!\uparrow)$ as $\texttt{minpre}\,(c)$. $\mathcal{T}$ is called *well-structured* if $\rightsquigarrow$ is *monotonic* wrt. $\preccurlyeq$, i.e. given three configurations $c_1, c_2, c_3 \in C$, if $c_1 \rightsquigarrow c_2$ and $c_1 \preccurlyeq c_3$ then there exists a fourth configuration $c_4 \in C$ such that $c_3 \rightsquigarrow c_4$ and $c_2 \preccurlyeq c_4$.

Given a configuration $c_{\texttt{target}} \in C$, the *coverability problem* asks whether there is a configuration $c' \succcurlyeq c_{\texttt{target}}$ reachable in $\mathcal{T}$. For the decidability of this problem the following conditions are sufficient: (i) For every two configurations c_1 and c_2 it is decidable whether $c_1 \preccurlyeq c_2$, (ii) for every $c \in C$, we can check whether $\{c\}\!\uparrow \cap C_{init} \neq \emptyset$, and (iii) for every $c \in C$, the set $\texttt{minpre}\,(c)$ is finite and computable.

The solution for the coverability problem of WSTS suggested in [2, 16] is based on a backward analysis approach. It is shown that starting from $U_0 := \{c_{\texttt{target}}\}$, the sequence $(U_i)_{i\geq 0}$ with $U_{i+1} := \texttt{min}\,(\texttt{Pre}\,(U_i)\!\uparrow \cup U_i\!\uparrow)$, for $i \geq 0$ reaches a fix point and is computable.

In the following, we instantiate the framework of WSTS to show the decidability of the state reachability problem for strongly bounded degenerative DRA, but first we need to introduce some notations.

Let k be a natural number, $D = \langle Q, q_0, M, X, \delta\rangle$ a degenerative DRA and $\texttt{target} \in Q$ a target state. Let $C_{init} = (C_{\texttt{init}}\,(D)\Diamond k)$ and $C = (C\,(D)\Diamond k)$. We use $\mathcal{T}^{\Diamond k}(D) = \left\langle C, C_{init}, \longrightarrow_D^{\Diamond k}\right\rangle$ to denote the corresponding k-strongly bounded transition system of D. We introduce the *reset prefix* transition relation $\rightsquigarrow := \xrightarrow{\texttt{reset}}{}_D^{*} \circ \longrightarrow_D^{\Diamond k}$. Note that the reflexive transitive closures of $\rightsquigarrow$ and $\longrightarrow_D^{\Diamond k}$ are identical. Thus, the state reachability of $\texttt{target}$ in $\left\langle C, C_{init}, \longrightarrow_D^{\Diamond k} \right\rangle$ is equivalent to its corresponding problem in $\langle C, C_{init}, \rightsquigarrow \rangle$. Next, we will prove the decidability of the latter problem.

We will show that $\langle C, C_{init}, \rightsquigarrow \rangle$ is a well-structured transition system. Let $c_{target} = \langle \{p\}, \texttt{s}, \texttt{r} \rangle$ be a configuration composed of a single process in state $\texttt{target}$ ($\texttt{s}(p) = \texttt{target}$) whose registers are empty ($\texttt{r}(p) = \bot_X$). We will define the well-quasi ordering on C in such a way that the upward closure of c_{target} consists of all configurations $c \in C$ with $\texttt{target} \in c$. Then, it is clear that the coverability of $c_{\texttt{target}}$ in $\langle C, C_{init}, \rightsquigarrow \rangle$ is equivalent to the reachability of $\texttt{target}$ in the same transition system.

In section 8.1, we define the well-quasi ordering $\preccurlyeq$ (Lemma 11) on C such that for every $c_1, c_2 \in C$ it is decidable whether $c_1 \preccurlyeq c_2$. The monotonicity of $\rightsquigarrow$ with respect to $\preccurlyeq$ is shown in section 8.2 (Lemma 12). The second sufficient condition for the decidability of the coverability problem, namely checking whether the upward closure of a configuration c contains an initial configuration, is trivial (we check whether c is an initial configuration). The last sufficient condition is shown by the following lemma:

▶ **Lemma 10.** *Given a configuration $c \in C$, we can effectively compute* $\texttt{minpre}(c)$.

Lemma 10, Lemma 11 and Lemma 12 show that coverability of c_{target} is decidable. Hence, the state reachability problem for strongly bounded degenerative DRA is decidable.

8.1 A well-quasi order on configurations

In this section, we define a well-quasi ordering $\preccurlyeq$ over the set C of configurations. Let us first introduce the notion of *subgraph* embedding. We use $\sqsubseteq_{sub}$ to denote the *subgraph* relation defined on graphs as follows: $\langle V_1, \Sigma_v, \Sigma_e, \lambda_1, E_1 \rangle \sqsubseteq_{sub} \langle V_2, \Sigma_v, \Sigma_e, \lambda_2, E_2 \rangle$ if there exists an injective mapping $t : V_1 \to V_2$ that is label and edge preserving, i.e. $\forall v, u \in V$ and $\forall a \in \Sigma_e$ we have $\lambda_1(v) = \lambda_2(t(v))$ and $\langle v, a, u \rangle \in E_1 \Rightarrow \langle t(v), a, t(u) \rangle \in E_2$. The subgraph relation over undirected (label-free) graphs are defined in a similar manner. We define the ordering $\preccurlyeq$ over the set of configurations as follows: Given two configurations $c_1 = \langle \texttt{procs}_1, \texttt{s}_1, \texttt{r}_1 \rangle$ and $c_2 = \langle \texttt{procs}_2, \texttt{s}_2, \texttt{r}_2 \rangle$, $c_1 \preccurlyeq c_2$ holds if $\texttt{enc}(c_1) \sqsubseteq_{sub} \texttt{enc}(c_2)$. Note that $c_1 \preccurlyeq c_2$ is equivalent to say that there exists an injective mapping $g : \texttt{procs}_1 \to \texttt{procs}_2$, such that (i) for every $p \in \texttt{procs}$, $\texttt{s}_1(p) = \texttt{s}_2(g(p))$ (ii) for every $p_1, p_2 \in \texttt{procs}_1$ and every $x \in X$, if $\texttt{r}_1(p_1)(x) = p_2$ then $\texttt{r}_2(g(p_1))(x) = g(p_2)$. It is easy to see that for two configurations c_1, c_2, we can check whether $c_1 \preccurlyeq c_2$.

▶ **Lemma 11.** *The relation $\preccurlyeq$ is a well-quasi ordering on C.*

8.2 Monotonicity

Let $c_1, c_2, c_3 \in C$ be three configurations such that $c_1 \preccurlyeq c_3$, i.e. the encoding of c_1 can be embedded in the encoding of c_3, and $c_1 \rightsquigarrow c_2$, i.e. there exist $c_1' \in C$ and $r \in \mathbb{N}$ such that $c_1 \xrightarrow{\texttt{reset}}{}_D^{r} c_1'$ and $c_1' \longrightarrow_D^{\Diamond k} c_2$. In order to prove $\rightsquigarrow$ monotonicity wrt. $\preccurlyeq$, we need to prove that there exists a fourth configuration $c_4 \in C$ such that $c_3 \rightsquigarrow c_4$ and $c_2 \preccurlyeq c_4$. To that end, we proceed by isolating the *sub configuration* c_{sub} induced by the embedding of c_1 into c_3 (see Figure 2). After a certain number r' (3 in Figure 2) of reset transitions $\xrightarrow{\texttt{reset}}_D$, one can obtain from c_3 a configuration c_3° composed of the disjoint union of the sub configuration c_{sub} and a set of *isolated* processes, i.e. processes whose registers are empty. As a consequence, diameters of c_3° and c_1 are equal. Furthermore, since c_{sub} is an embedding of c_1 into c_3°, and since $\oslash(\texttt{closure}(\texttt{enc}(c_3^\circ))) = \oslash(\texttt{closure}(\texttt{enc}(c_1)))$, c_3° can perform the

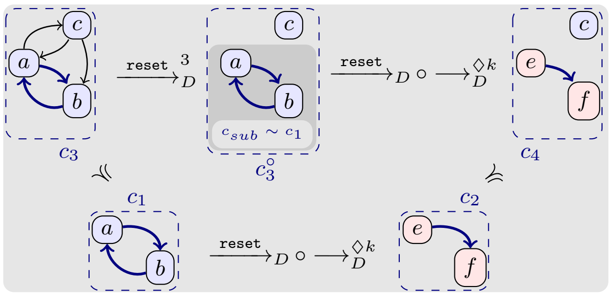

c
a
b
c_3
$\xrightarrow{\texttt{reset}}{}_D^{3}$
$c_{sub} \sim c_1$
c_3°
$\xrightarrow{\texttt{reset}}{}_D \circ \longrightarrow_D^{\Diamond k}$
e
f
c_4
$\preccurlyeq$
c_1
c_2

Figure 2 Monotonicity and reset transitions.

same transition as c_1 did in order to get to c_2 without violating the bound k. Thus, after two consecutive transitions whose composition ($\xrightarrow{\texttt{reset}}{}_D^{r'} \circ \xrightarrow{\texttt{reset}}{}_D^{r} \circ \longrightarrow_D^{\Diamond k} = \xrightarrow{\texttt{reset}}{}_D^{r'+r} \circ \longrightarrow_D^{\Diamond k}$) is a $\rightsquigarrow$ -transition, c_3 can reach a configuration where c_2 can be embedded.

▶ **Lemma 12.** *The transition relation* $\rightsquigarrow$ *is monotonic w.r.t.* $\preccurlyeq$.

9 Conclusion and Future Directions

We have presented the first work addressing the state reachability problem for Dra. We have shown that this problem is undecidable and that this undecidability holds even if we restrict the analysis to the case where transitions are only allowed between (strongly) bounded configurations (i. e., simple paths of the underlying (undirected) graph are bounded by some constant), unlike the case of Ad-hoc networks [10, 11, 13, 1]. Our main goal was to identify subclasses of Dra for which the reachability problem is decidable. To that end, we have introduced degenerative Dra for which any register can be reset in a non-deterministic manner. We have shown that the sets of reachable states of a Dra and its degenerative counterpart are identical. Moreover, we have shown that the reachability problem for degenerative Dra becomes decidable but nonprimitive recursive when we restrict the analysis to strongly bounded configurations. Furthermore, we have considered (strongly) safe Dra where we assume that all reachable configurations are (strongly) bounded. We have shown that the state reachability problem is decidable for strongly safe Dra.

To the best of our knowledge these are the first results concerning the verification of dynamic register automata. While the communication in Dra is rendezvous based, the automata models considered in [8] and [7] use asynchronous communication through unbounded channels. It is well-known that, even for finitely many processes communicating through unbounded perfect FIFO channels, most of the interesting verification questions are undecidable [9]. A possible direction of further research would be to investigate whether our decidability result carries over to the case of asynchronous communication through *“well-structured”* channels (e.g., bounded, lossy, unordered).

Acknowledgements. Authors would like to thank Benedikt Bollig, Thomas Schwentick and Thomas Zeume for their careful reading of the paper and their helpful comments.

References

1 Parosh Aziz Abdula, Mohamed Faouzi Atig, and Othmane Rezine. Verification of directed acyclic ad hoc networks. In *FMOODS/FORTE*, pages 193–208, 2013.

2 P. A. Abdulla, K. Cerans, B. Jonsson, and Y. K. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321. IEEE Computer Society, 1996.

3 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996.

4 Bharat Adsul, Madhavan Mukund, K. Narayan Kumar, and Vasumathi Narayanan. Causal closure for MSC languages. In *FSTTCS*, volume 3821 of *LNCS*, pages 335–347. Springer, 2005.

5 Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and verification of MSC graphs. *Theor. Comput. Sci.*, 331(1):97–114, 2005.

6 Michael Benedikt, Clemens Ley, and Gabriele Puppis. Automata vs. logics on data words. In *CSL*, volume 6247 of *LNCS*, pages 110–124. Springer, 2010.

7 Benedikt Bollig, Aiswarya Cyriac, Loïc Hélouët, Ahmet Kara, and Thomas Schwentick. Dynamic communicating automata and branching high-level MSCs. In *LATA*, volume 7810 of *LNCS*. Springer, 2013.

8 Benedikt Bollig and Loïc Hélouët. Realizability of dynamic MSC languages. In *CSR*, volume 6072 of *LNCS*. Springer, 2010.

9 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

10 G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR'10*, volume 6269 of *LNCS*. Springer, 2010.

11 G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FoSSaCS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.

12 Giorgio Delzanno, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of broadcast networks of register automata. In *RP*, volume 8169 of *LNCS*. Springer, 2013.

13 Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS*, volume 18 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

14 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. In *LICS*, pages 17–26. IEEE Computer Society, 2006.

15 D. Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012.

16 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

17 Blaise Genest, Dietrich Kuske, and Anca Muscholl. A kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.

18 Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, Milind A. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Inf. Comput.*, 202(1):1–38, 2005.

19 M. Jurdzinski and R. Lazic. Alternation-free modal mu-calculus for data trees. In *LICS*, pages 131–140. IEEE Computer Society, 2007.

20 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

21 Ranko Lazic. Safely freezing LTL. In *FSTTCS*, volume 4337 of *LNCS*, pages 381–392. Springer, 2006.

22 Anca Muscholl and Doron Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *MFCS*, volume 1672 of *LNCS*, pages 81–91. Springer, 1999.

23 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

24 Hiroshi Sakamoto and Daisuke Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000.

25 P. Schnoebelen. Revisiting ackermann-hardness for lossy counter machines and reset petri nets. In *MFCS*, volume 6281 of *LNCS*, pages 616–628. Springer, 2010.

26 Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. Query-based model checking of ad hoc network protocols. In *CONCUR*, volume 5710 of *LNCS*, pages 603–619. Springer, 2009.

27 Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. A process calculus for mobile ad hoc networks. *Sci. Comput. Program.*, 75(6):440–469, 2010.

28 Nikos Tzevelekos. Fresh-register automata. In *POPL*. ACM, 2011.