31st International Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2011, December 12–14, 2011, Mumbai, India

Edited by

Supratik Chakraborty

Amit Kumar

Editors

Supratik Chakraborty
Dept. of Computer Science & Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai 400076
India
supratik@cse.iitb.ac.in

Amit Kumar
Dept. of Computer Science & Engineering
Indian Institute of Technology, Delhi
Hauz Khas, New Delhi 110016
India
amitk@cse.iitd.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

ISBN 978-3-939897-34-7

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/978-3-939897-34-7.

Publication date
December, 2011

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.2011.i

ISBN 978-3-939897-34-7 **ISSN 1868-8969** **www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

www.dagstuhl.de/lipics

Contents

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: S. Chakraborty, A. Kumar
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session 2A

Session 2B

Session 3A

Session 3B

Session 4A

Session 4B

Session 5A

Session 5B

Session 6

Session 7A

Session 7B

Preface

The 31^{st} international conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011) was held at Indian Institute of Technology, Bombay from December 12 to December 14, 2011. This proceedings volume contains contributed papers and (extended) abstracts of invited talks presented at the conference.

FSTTCS is the annual flagship conference of the Indian Association for Research in Computing Science (IARCS). Over the past 31 years, the conference has consistently attracted top-quality submissions, and the legacy continued this year as well. There were a total of 116 submissions, with contributing authors from 32 countries, spanning 6 continents. Most of these were high-quality papers, and the competition for making it to the final list of accepted papers was very stiff. We wish to thank all authors of submitted papers for helping make FSTTCS such a competitive conference.

The program committee, in which we had the privilege of working with 30 distinguished colleagues from 13 countries, spent almost two months reviewing and discussing in great detail the strengths and weaknesses of submitted papers. The committee also sought help from 243 external reviewers, and was able to obtain a total of 447 reviews for the contributed papers. All papers within the scope of the conference had at least 3 expert reviews, and all but a few papers were evaluated by 4 independent reviewers. Multiple rounds of elimination and intense discussions and debates followed to help us narrow down the set of papers to be accepted. A total of 37 papers were finally selected for presentation at the conference. Several very good papers had to be left out of this list owing to stiff competition. Our thanks to all members of the program committee for lending their valuable time, expertise and acumen for this significant task. Without their inputs, hard work and detailed critical comments, it would not have been possible to maintain the high standards that FSTTCS has set for itself over the years. We would also like to take this opportunity to thank all external reviewers for their detailed comments and critical assessment of papers.

Six eminent computer scientists readily agreed to travel from different parts of the world to deliver invited talks at the conference. We would like to thank Susanne Albers, Phokion G. Kolaitis, John C. Mitchell, Madhu Sudan, Moshe Y. Vardi and Umesh V. Vazirani for enriching the conference with their invited talks and with their participation. It has been our privilege to have them as part of FSTTCS 2011.

Paper submission, reviewing and program committee discussion for the conference were managed using EasyChair. We wish to thank Andrei Voronkov and the entire EasyChair team for making this excellent service available to us, and for ensuring that the reviewing, discussion and paper management processes went smoothly.

We would like to thank IIT Bombay and its administration for allowing us to use the facilities of the institute for hosting this prestigious conference in its premises. We would like to thank staff, students, post-doctoral research scholars and faculty colleagues of the Dept. of Computer Science and Engineering at IIT Bombay for pitching in myriad ways to help make the organization of the conference a success. Our special thanks go to Hrishikesh Karmarkar for ensuring that several organizational issues that threatened to spiral out of control, were eventually tamed in time. We would like to thank our student system administrators, Prajish Prasad and Jagadish M., for helping with networking and all computer system related issues. The team of project "ekalavya" at IIT Bombay, under the guidance of Prof. D.B. Phatak, readily agreed to record the proceedings of the conference, and make the recordings available in open source under a Creative Commons license. We would like to thank the

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: S. Chakraborty, A. Kumar
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

entire "ekalavya" team for lending us such a big helping hand. It is also our pleasure to acknowledge the financial support we received from IARCS and from four additional sponsors for FSTTCS 2011. These additional sponsors were: Google India Pvt. Ltd., Dept. of Computer Science and Engineering at IIT Bombay, Tata Consultancy Services and IBM India Pvt. Ltd. The institutional members of IARCS, whose contributions help support various activites of IARCS, including FSTTCS, are Geodesic Ltd., Microsoft Research India and Sasken Communication Technologies Ltd.

In addition to the technical program of the main conference, two satellite events were co-located with FSTTCS 2011. These included a pre-conference workshop on Finite and Algorithmic Model Theory and a post-conference workshop on Breakthroughs in Theoretical Computer Science. The workshop on Finite and Algorithmic Model Theory featured Anuj Dawar, Phokion G. Kolaitis, Emanuel Kieroński, Dietmar Berwanger, Benjamin Rossman and R. Ramanujam as speakers. The workshop on Breakthroughs in Theoretical Computer Science had Manindra Agrawal, Naveen Garg, Sanjeev Khanna, Aleksander Mądry, Jaikumar Radhakrishnan, Madhu Sudan, Umesh V. Vazirani, Amit Kumar and Nisheeth Vishnoi as speakers. Our thanks to all invited speakers of the two workshops for adding to the richness and diversity of the technical program.

As in the past few years, the proceedings of FSTTCS 2011 is being 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. We wish to thank the editorial board of LIPIcs for agreeing to publish the current proceedings as a LIPIcs volume. Our special thanks to Marc Herbstritt for answering our deluge of questions pertaining to publishing and editing issues clearly and promptly. We gratefully acknowledge his help and the help of the entire team in the LIPIcs editorial office in preparing the final version of the proceedings.

Supratik Chakraborty and Amit Kumar
Mumbai, December 2011

Conference Organization

Program Chairs

Supratik Chakraborty (IIT Bombay, India)
Amit Kumar (IIT Delhi, India)

Program Committee

Luca Aceto (Reykjavik U., Iceland)
Yossi Azar (Tel-Aviv U., Israel)
Mikołaj Bojańczyk (Warsaw U., Poland)
Krishnendu Chatterjee (IST, Austria)
Shuchi Chawla (U. Wisconsin-Madison, USA)
Anuj Dawar (U. of Cambridge, UK)
Stephanie Delaune (LSV-ENS Cachan, France)
Benjamin Doerr (MPII, Saarbrücken, Germany)
Zoltań Ésik (U. of Szeged, Hungary)
Friedrich Eisenbrand (EPFL, Switzerland)
Javier Esparza (TU München, Germany)
Martin Fränzle (U. Oldenburg, Germany)
Sariel Har-Peled (UIUC, USA)
Prahladh Harsha (TIFR, India)
Holger Hermanns (Saarland U., Germany)
Radha Jagadeesan (DePaul U., USA)
Ragesh Jaiswal (Columbia U., USA)
Sanjeev Khanna (U. of Pennsylvania, USA)
K. Narayan Kumar (CMI, India)
Kim G. Larsen (Aalborg U., Denmark)
Ashwin Nayak (U. of Waterloo, Canada)
Madhusudan Parthasarathy (UIUC, USA)
Seth Pettie (U. Michigan, Ann Arbor, USA)
S. Ramesh (GM ISL, India)
Saket Saurabh (IMSc, India)
Anil Seth (IIT Kanpur, India)
Aravind Srinivasan (UMCP, USA)
Martin Strauss (U. Michigan, Ann Arbor, USA)
Ashish Tiwari (SRI International, USA)
Nisheeth K. Vishnoi (Microsoft Research, India)

Organizing Committee

Bharat Adsul (IIT Bombay, India)
A. K. Bhattacharjee (BARC, India)
Supratik Chakraborty (IIT Bombay, India)
Ashish Chiplunkar (IIT Bombay, India)
Ayush Choure (IIT Bombay, India)
Jugal Garg (IIT Bombay, India)
Pravin Jadhav (IIT Bombay, India)
Hrishikesh Karmarkar (IIT Bombay, India)
S. Krishna (IIT Bombay, India)
Nutan Limaye (IIT Bombay, India)
Jagadish M. (IIT Bombay, India)
Jinesh Machchhar (IIT Bombay, India)
Ruta Mehta (IIT Bombay, India)
Soumitra Pal (IIT Bombay, India)
Prajish Prasad (IIT Bombay, India)
Abhiram Ranade (IIT Bombay, India)
Abhisekh Sankaran (IIT Bombay, India)
Shetal Shah (IIT Bombay, India)
Chandrakant Talekar (IIT Bombay, India)

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: S. Chakraborty, A. Kumar

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Fidaa Abed
Bharat Adsul
Shipra Agrawal
Susanne Albers
Eric Allender
Antonios Antoniadis
Takahito Aoto
Mohamed Faouzi Atig
Christel Baier
Sandie Balaguer
Maria-Florina Balcan
Nikhil Bansal
Vince Barany
Siddharth Barman
Djamal Belazzougui
Henrik Björklund
Udi Boker
Benedikt Bollig
Remi Bonnet
Patricia Bouyer
Tomas Brazdil
Vaclav Brozek
Peter Bulychev
Guillaume Burel
Danny Bøgsted Poulsen
Iliano Cervesato
Rohit Chadha
Sourav Chakraborty
Timothy Chan
Rajesh Chitnis
Marek Chrobak
Matteo Cimini
Anthony Coadou
Thomas Colcombet
Kevin Compton
Bruno Courcelle
Dipankar Das
Alexandre David
David De Frutos
Julie De Pril
Stephane Demri
Amit Deshpande
Henning Dierks
Dino Distefano
Laurent Doyen
Manfred Droste
Andreas Eggers
Christian Eisentraut
Thomas Erlebach
Yuri Faenza
Uli Fahrenberg
Tomás Feder
José Luiz Fiadeiro
Nathanael Fijalkow
Bernd Finkbeiner
Dimitris Fotakis
Mahmoud Fouz
Fabrizio Frati
Kimmo Fredriksson
Dominik D. Freydenberger
Sorelle Friedler
Sibylle Fröschle
Stanley Fung
Stefan Funke
Pierre Ganty
Naveen Garg
Simon Gay
Zsolt Gazdag
Blaise Genest
Amelie Gheerbrant
Hugo Gimbert
Petr Golovach
Manoj Gopalkrishnan
Eugen-Ioan Goriac
Sathish Govindarajan
Fabrizio Grandoni
Ankit Gupta
Anupam Gupta
Annegret Habel
Michel Hack
Avinatan Hassidim
Michael Hoffmann
Piotr Hofman
Nicolai Hähnle

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: S. Chakraborty, A. Kumar
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Hans Hüttel
Szabolcs Iván
Petr Jančar
Bart Jansen
David N. Jansen
Gwenaël Joret
Marcin Jurdziński
Aditya Kanade
Haim Kaplan
Prateek Karandikar
Hrishikesh Karmarkar
Ken-Ichi Kawarabayashi
Neeraj Kayal
Andrew King
Shiva Kintali
Christian Knauer
Barbara König
Naoki Kobayashi
Eryk Kopczyński
Vijay Anand Korthikanti
Stefan Kratsch
Jörg Kreiker
Jan Kretinsky
Antonin Kucera
Manfred Kufleitner
Orna Kupferman
Martin Kutrib
Markus Latte
Ranko Lazić
Lap-Kei Lee
Axel Legay
Henry C.M. Leung
Clemens Ley
Xueliang Li
Daniel R. Licata
Nutan Limaye
Anthony Widjaja Lin
Andreas Lochbihler
Kamal Lodaya
Satyanarayana Lokam
Daniel Lokshtanov
Alejandro Lopez-Ortiz
Michael Luttenberger
Christof Löding
Meena Mahajan
Michael Mahoney
Andreas Maletti
Sebastian Maneth
Amaldev Manuel
Nicolas Markey
Dániel Marx
Ruta Mehta
Raghu Meka
Paul-André Melliès
Stefan Mengel
Massimo Merro
Roland Meyer
Ulrich Meyer
Jakub Michaliszyn
Zoltan Miklos
Sounaka Mishra
Swarup Mohalik
Carsten Moldenhauer
Apurva Mudgal
Madhavan Mukund
N. Raja
Rajagopal Nagarajan
N.S. Narayanaswamy
Martin Niemeier
Jan Obdrazalek
Martin Otto
Youssouf Oualhadj
Scott Owens
Konstantinos Panagiotou
Paritosh Pandya
Gennaro Parlato
Daniel Paulusma
Mikkel L. Pedersen
Gustavo Petri
Nir Piterman
Thomas Place
Libor Polak
Ely Porat
M. Praveen
David Pritchard
Kirk Pruhs
Arjun R.
R. Ramanujam
Arash Rafiey
Willard Rafnsson

C.R. Ramakrishnan
Venkatesh Raman
Jean-Francois Raskin
Michael Rink
Thomas Rothvoss
Sambuddha Roy
Krishna S.
Yogish Sabharwal
Sushant Sachdeva
Prakash Saivasan
Jacques Sakarovitch
Rishi Saket
Sylvain Salvati
Prahladavaradan Sampath
Abhisekh Sankaran
Ocan Sankur
Luigi Santocanale
Jayalal Sarma M.N.
Srinivasa Rao Satti
Zdenek Sawa
Sven Schewe
Maximilian Schlund
Henning Schnoor
Grant Schoenebeck
Thomas Schwentick
Sandeep Sen
Olivier Serre
Farhad Shahrokhi
K.C. Shashidhar
Sarai Sheinvald
Sunil Simon
Randy Smith
Pawel Sobocinski
Jiri Srba
Srikanth Srinivasan
Piyush Srivastava
Sam Staton
S.P. Suresh
Ola Svensson
Mani Swaminathan
Chaitanya Swamy
Stefan Szeider
Robert Tarjan
Tino Teige
Dimitrios Thilikos
Wolfgang Thomas
Claus Thrane
Tobe Toben
Akihiko Tozawa
Madhur Tulsiani
Seeun Umboh
Michael Ummels
Pawel Urzyczyn
Jouko Väänänen
Sándor Vágvölgyi
Vincent Van Oostrom
Mahesh Viswanathan
Adrian Vladu
Heiko Vogler
Heribert Vollmer
Björn Wachter
William Wadge
Magnus Wahlström
Oren Weimann
Bernd Westphal
Raphael Yuster
Lijun Zhang
Shengyu Zhang
Martin Zimmermann

Author Index

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. xvi–xvii
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Energy-Efficient Algorithms

Susanne Albers*[1]

1 **Department of Computer Science**
Humboldt-Universität zu Berlin, Germany
albers@informatik.hu-berlin.de

Abstract

This presentation surveys algorithmic techniques for energy savings. We address power-down as well as dynamic speed scaling mechanisms.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases energy efficiency, power-down mechanisms, dynamic speed scaling, offline algorithm, online algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.1

Summary

Algorithmic techniques for energy savings have received considerable research interest over the past years. Power dissipation is critical in portable, battery operated devices where the amount of available energy is severely limited. Moreover, energy consumption is a concern in desktop computers and servers as electricity costs impose a substantial strain on the budget of computing and data centers. Last but not least, power dissipation causes thermal problems. Most of the consumed energy is eventually converted into heat, resulting in wear and reduced reliability of hardware components.

Algorithmic work on energy conservation focuses mostly on the system and device level: How can we save energy in a given computational device? The following two mechanisms have been studied extensively.

- *Power-down mechanisms*: When a system is idle, it can be transitioned into low-power standby or sleep states. This technique is well-known and widely used. The goal is to determine good shutdown times subject to the constraint that a transition back to the active mode requires extra energy.
- *Dynamic speed scaling*: Microprocessors currently sold by chip makers such as AMD and Intel are able to operate at variable speed. The higher the speed, the higher the power consumption is. The goal is to save energy by applying the full speed/frequency spectrum of a processor and utilizing low speeds whenever possible.

In this talk we will review the above techniques along with some fundamental results. Most of the presentation will be devoted to dynamic speed scaling. We concentrate on a basic processor scheduling problem introduced in a seminal paper by Yao, Demers and Shenker [3]. Here a set of jobs, each specified by a release time, a deadline and a processing volume, has to be scheduled on a variable-speed processor so as to minimize energy consumption. Yao et al. devised a polynomial time offline algorithm as well as two online algorithms. We study settings with parallel processors and present efficient offline and online algorithms developed in [2]. Moreover, we investigate a setting where a variable-speed processor is equipped with

* Work supported by the German Research Foundation.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 1–2

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a sleep state. We show how to integrate power-down and speed scaling mechanisms and give efficient offline approximation algorithms [1].

References

1 S. Albers and A. Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *Proc. 23rd ACM-SIAM Symposium on Discrete Algorithms,* 2012.

2 S. Albers, A. Antoniadis and G. Greiner. On multi-processor speed scaling with migration. *Proc. 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures,* 2011.

3 F.F. Yao. A.J. Demers and S. Shenker. A scheduling model for reduced CPU energy. *Proc. 36th IEEE Symposium on Foundations of Computer Science*, 374–382, 1995.

Constraints, Graphs, Algebra, Logic, and Complexity*

Moshe Y. Vardi[1]

1 **Department of Computer Science**
Rice University, Houston, TX 77005, USA
vardi@cs.rice.edu.com

Abstract

A large class of problems in AI and other areas of computer science can be viewed as constraint-satisfaction problems. This includes problems in database query optimization, machine vision, belief maintenance, scheduling, temporal reasoning, type reconstruction, graph theory, and satisfiability. All of these problems can be recast as questions regarding the existence of homomorphisms between two directed graphs. It is well-known that the constraint-satisfaction problem is NP-complete. This motivated an extensive research program into identify tractable cases of constraint satisfaction.

This research proceeds along two major lines. The first line of research focuses on non-uniform constraint satisfaction, where the target graph is fixed. The goal is to identify those target graphs that give rise to a tractable constraint-satisfaction problem. The second line of research focuses on identifying large classes of source graphs for which constraint-satisfaction is tractable. We show in how tools from graph theory, universal algebra, logic, and complexity theory, shed light on the tractability of constraint satisfaction.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases constraint satisfaction, NP completeness, dichotomy

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.3

References

1 P.G. Kolaitis and M.Y. Vardi. A logical approach to constraint satisfaction. In *Complexity of Constraints*, Lecture Notes in Computer Science 5250, pp. 125–155, Springer, 2008.

* Work supported in part by NSF grants CCF-0728882, and CNS 1049862, by BSF grant 9800096, and by gift from Intel.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 3–3

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Physical limits of Communication

Madhu Sudan[1]

1 **Microsoft Research New England, One Memorial Drive, Cambridge, MA 02142, USA.** madhu@mit.edu

Abstract

We describe recent work with Sanjeev Khanna (U. Penn.) where we explore potential axioms about the mechanics of information transmission with a view to understanding whether continuous signals can carry more information than analog signals.

1998 ACM Subject Classification H.1.1 Systems and Information Theory

Keywords and phrases Analog signals, information capacity, delays

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.4

1 Summary

Year after year, we have seen explosions in the amount of information that we can store on storage devices, and the speed at which we can ship this information around the universe. Are we close to reaching limits imposed by the physics? What are these limits and what axioms do they come from? The study of such questions has primarily been the domain of signal processing and information theory. In this talk we will speak about joint work with Sanjeev Khanna, in which we explore such questions with some discrete and probabilistic modeling.

The fundamental underlying issue is that digital communication is implemented on physical objects. Classical models of the communication power of this physical layer study the information carrying power of "continuous time signals", i.e., real-valued functions over a continuous interval of time. In the absence of sources of uncertainty such signals have "infinite" information carrying capacity for two reasons: (1) At each instant of time, the signal has infinitely many different values; and (2) There are infinitely many instances of time (even in a bounded interval).

The former issue was resolved quite convincingly in the work of Shannon, who said that if there is a additive error at each instant of time (where the error could be a simple Gaussian random variable), then the information carrying power of a signal bounded in energy is a finite number of bits. The latter issue unfortunately appears to be a bit more subtle. Classical signal processing tends to deal with this issue by first looking at the Fourier spectrum of the signal being transmitted, and then placing some restrictions on these. Converting back to time domain, these restrictions miraculously say it is sufficient to sample the signal at discrete time intervals. But what are these restrictions, where did they come from, and are they really distinct from the assumption that the signal should be inferrable from its values at a discrete set of sample points? In this talk we discuss such questions, and then contrast with the more discrete and probabilistic models analyzed in our work [1].

In our work we ask what happens if nature provides us with a collection of N particles, each one of which is capable of transmitting one bit somewhat unreliably. In particular we study the case where this unreliability comes in two forms: *error*, where the bit carried by a particle may flip during transmission, and *delay*, where the particle's arrival time at a

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 4–5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

destination may not correspond exactly to its departure time. In particular we let the delay be a random variable with a variance of one unit of time, and study how much information can the sender communicate in T units of time, The natural method to deal with this uncertain delay would be to transmit roughly N/T particles every, say, two units of time and then the delay variance of one unit of time would not lead to interference from particles transmitted at different time slots. But this would lead to a rate of information transmission of only $T \log(N/T)$ bits. A more "continuous" transmission protocol might allow particles to be transmitted at any point of time in the interval $[0, T]$. Would this enhance the capacity? Surprisingly our (simple) analysis shows that the capacity does grow with such schemes. In particular one can transmit as $T \cdot (N/T)^{\epsilon}$ bits, for some positive ϵ, in T units of time in such settings. Thus we find that using the continuum of time does enhance capacity, up to physical limits imposed by the particular nature of matter.

References

1 Sanjeev Khanna and Madhu Sudan. Delays and the capacity of continuous-time channels. *CoRR*, abs/1105.3425, 2011.

A Domain-Specific Language for Computing on Encrypted Data

Alex Bain, John Mitchell, Rahul Sharma, Deian Stefan and Joe Zimmerman

Stanford University, Stanford, CA

Abstract

In cloud computing, a client may request computation on confidential data that is sent to untrusted servers. While homomorphic encryption and secure multiparty computation provide building blocks for secure computation, software must be properly structured to preserve confidentiality. Using a general definition of *secure execution platform*, we propose a single Haskell-based domain-specific language for cryptographic cloud computing and prove correctness and confidentiality for two representative and distinctly different implementations of the same programming language. The secret sharing execution platform provides information-theoretic security against colluding servers. The homomorphic encryption execution platform requires only one server, but has limited efficiency, and provides secrecy against a computationally-bounded adversary. Experiments with our implementation suggest promising computational feasibility, as cryptography improves, and show how code can be developed uniformly for a variety of secure cloud platforms, without explicitly programming separate clients and servers.

1998 ACM Subject Classification D.3.3 Language Constructs and Features

Keywords and phrases Domain-Specific Language, Secret Sharing, Homomorphic Encryption

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.6

1 Introduction

Recent advances in secure multiparty computation and homomorphic encryption promise a wide range of new applications. In particular, it is cryptographically possible to protect data in the cloud from the servers manipulating it, subject to varying threat models. However, the practical widespread use of these cryptographic techniques requires a suitable software development, testing, and deployment infrastructure.

In this paper, we present the design, foundational analysis, implementation, and performance benchmarks for an initial embedded domain-specific language (EDSL) that allows programmers to develop code that can be run on different secure execution platforms with different security guarantees. Figure 1 shows how our separation of programming environment from cryptographically secure execution platforms can be used to delay deployment decisions or run the same code on different platforms.

While homomorphic encryption and secure multiparty computation are based on different cryptographic insights and constructions, there is a surprising structural similarity among them that we express in our definition of *secure execution platform*. This definition allows us to develop a single set of additional definitions, theorems, and proofs that are applicable to many platforms. In particular, we prove functional correctness and confidentiality, for an honest-but-curious adversary, across relevant platforms. We then show that fully homomorphic encryption satisfies our definition, as does a specific secret-sharing scheme, subject to assumptions on the number of potentially colluding servers. Moreover,

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 6–24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

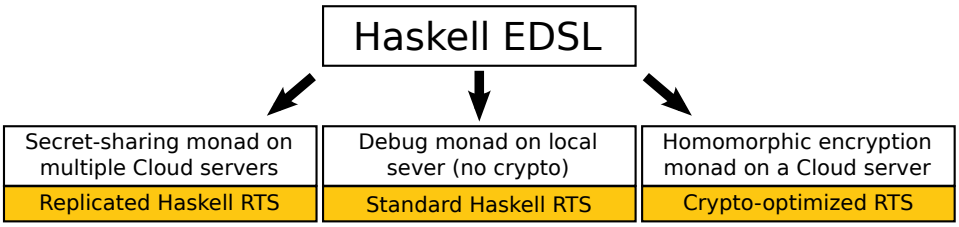


Figure 1 Multiple deployment options using different runtime systems (RTS)

our definition of secure execution platform is parameterized over the set of primitive operations on secret values, so that our language and our theoretical guarantees are applicable to partially homomorphic schemes, when they support the operations actually used in the program code. Our correctness theorems show equivalence with a reference implementation and therefore imply output equivalence for alternative secure execution platforms.

Our embedded domain-specific language is implemented as a Haskell library, rather than as a completely new language, so that developers can use existing and carefully engineered Haskell development tools, compilers, and run-time systems. Programmers also have the benefit of sophisticated type-checking and general programming features of Haskell because we rely only on the Haskell type discipline, not *ad hoc* code restrictions. Further, we use the Haskell type system to impose an information-flow discipline that is critical to preserving confidentiality against cloud servers that could otherwise leak information through control-flow analysis or other forms of program monitoring. Our Haskell implementation also provides flexible data structures, since our information-flow constraints make secrecy-preserving operations on such such structures possible.

As a working example, we consider the Generalized Millionaires' Problem: given a number of millionaires, request their net worth, identify the richest millionaire, and, finally, notify each one of their status without revealing their net worth. Figure 2 shows an example implementation, that highlights several key aspects of our Haskell EDSL. First, our language provides various primitives such as `withUsers`, `uRead`, and `reveal` that are respectively used to apply a function (e.g., `readWorth`) to each connected user, read a secret input from the user, and reveal (decrypt) a secret value. Second, the DSL embedding allows the programmer to use existing Haskell features including higher-order functions, abstract data types, general recursion, etc. An example use of recursion in our example is `foldlM1` which, with `maxWorth`, is used to find the richest millionaire. Finally, compared to languages with similar goals (e.g., SMCL [24]), where a programmer is required to write separate client and server code, using our EDSL, a programmer need only write a single program; we eliminate the client/server code separation by providing a simple runtime system that directs all parties.

```
generalizedMillionaires = do
  -- Read worth from all users:
  allWorth ← withUsers readWorth
  -- Find richest user and her worth
  (richest, worth) ← foldlM1 maxWorth allWorth
  -- Notify the users of the status:
  richestUser ← reveal richest
  withUsers_ (λu →
    if u == richestUser
      then uPutStrLn u "You are the richest!"
      else uPutStrLn u "Keep working!")

  where readWorth u =  do
          w ← uRead u
          return (hide u, w)
        maxWorth (u1,w1) (u2, w2) = do
          b ← (w1 .> w2)
          sif b sthen (u1,w1) selse (u2, w2)
```

Figure 2 Generalized Millionaires' Problem

We describe a Haskell implementation of secure execution platforms based on both secret

sharing and fully homomorphic encryption, both using SSL network communication between clients and any number of servers. Our implementation effort produced 2500 lines of Haskell and 650 lines of C/C++ code. We developed sample applications and measured performance on benchmarks, as reported in Section 4.3. Because our implementation is packaged in the form of Haskell libraries, other researchers could use our libraries to implement other programming paradigms over the same forms of cryptographic primitives. Conversely, we could target our language to other run-time systems such as SMCR [24], for programmers only interested in that execution paradigm.

The contributions of this work include:

- We leverage the similarity between secure multiparty computation and homomorphic encryption, as captured in a precise definition of *secure execution platform*.
- We design, implement, and test an embedded DSL that allows programmers to develop code that runs on any secure execution platform supporting the operations used in the code. We avoid *ad hoc* language restrictions by relying only on the Haskell type system for information flow properties and other constraints.
- We prove general functional correctness and security theorems, beyond previous work on related languages for secure multiparty computation (SMC [27], Fairplay [22], SIMAP [4, 24] and VIFF [8]).
- We develop and evaluate distributed secret sharing and homomorphic encryption execution platforms, using SSL network communication, implemented in Haskell.

Although we develop our results using the commonly used honest-but-curious adversary model, there are established methods for assuring integrity, using commitments and zero-knowledge techniques [16]. Moreover, since these add communication and computation overhead, we can also consider the possibility of using techniques from [23] in future work. These methods employ *computational* commitment and proofs of knowledge to provide computations on ciphertexts with verifiable integrity and smaller overhead. While we focus on data confidentiality, we can also protect confidential algorithms by considering code as input data to an interpreter (or "universal Turing machine").

2 Background

We propose a domain-specific programming language (DSL) embedded in Haskell, drawing on previous languages (e.g., Cryptol), use of monads for cryptographic computation, and other works on programmable secure multiparty computation (e.g., Fairplay [22], SIMAP [4, 24]). In this section, we introduce Haskell, and review secure multiparty computation and homomorphic encryption.

Haskell and EDLs Haskell is a widely used host language for EDSLs [17]. The language offers a strong, static type system that includes parametric and ad-hoc polymorphism (via type classes); first-class monads, with convenient syntactic sugar; and, the IO monad, strictly separating pure from impure computations. Haskell's type classes, lazy evaluation strategy (i.e., expressions are evaluated only when their values are needed), and support for monads makes it easy to define new data structures, syntactic extensions, and control structures—features commonly desired when embedding DSLs.

The main Haskell constructs used in embedding our DSL are monads and type classes. A monad M provides a type constructor and related operations that obey several laws. Specifically, if M is a monad and α

```
class Monad M where
  return :: α → Mα
  (>>=) :: Mα → (α → Mβ) → Mβ
```

Figure 3 Monad operations

is an arbitrary type, then $M\alpha$ is a type with operations `return`, and $\gg\!=$ (pronounced "bind"), whose types are shown in Figure 3. As shown in the figure, Haskell provides support for monads through the `Monad` *type class*. Type classes provide a method of associating a collection of operations with a type or type constructor. Programmers, then, declare *instances* of a given type class by naming the type or type constructor and providing implementations of all required operations. As explained later, type classes are also useful in 'overloading' arithmetic operations over secret and public data.

Homomorphic encryption A *homomorphic encryption scheme* $\langle \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval} \rangle$ consists of a key generation algorithm, encryption and decryption algorithms, and an evaluation function that evaluates a function $f \in \mathcal{F}$ on one encrypted value to produce another. More specifically, if $c = \mathsf{Enc}(\mathrm{pk}, m)$ then $\mathsf{Eval}(\mathrm{pk}, c, f) = \mathsf{Enc}\big(\mathrm{pk},\ f(m)\big)$ for every $f \in \mathcal{F}$, where $\mathcal{F} \subseteq (\text{Plaintext} \to \text{Plaintext})$ is some set of functions on plaintexts. As stated here, $\mathcal{F}$ is a set of unary functions; however, we consider the more general case where each function has a specific arity and type. We say that the scheme is *homomorphic with respect to the set $\mathcal{F}$ of functions.*

While some homomorphic encryption schemes [6, 15, 20] are homomorphic with respect to a restricted class of functions, such as the set of quadratic multivariate polynomials or the set of shallow branching programs, recent research has produced an encryption scheme that is *fully homomorphic*, i.e., homomorphic with respect to all functions of polynomial complexity [11, 12, 28, 29]. Since this work has generated substantial interest, there is a rapidly growing set of fully homomorphic constructions. However, for efficiency reasons we remain interested in partially homomorphic schemes as well. Moreover, for any given program, it is only necessary to use a form of homomorphic encryption that is sufficient for the functions used by that program.

Secure multiparty computation Another approach to computing on ciphertexts makes use of generic 2-party or multi-party secure computation [30, 21, 2, 18, 7, 23, 19, 9, 1], in which the client, who has the plaintext x, communicates through some protocol with the server(s), who have the function f to be computed on x. The standard conditions for secure multiparty computation guarantee that the client only learns $f(x)$ and the server learns nothing about x.

In Shamir secret sharing and the multi-party computation algorithm based on it (see [10]), a client C shares a secret value a_0 from a finite field F among N other parties that we will refer to as *servers*. In an (N, k) secret sharing scheme, N servers can jointly perform computations on a_0 and other shared secrets, such that at least k of the N servers must cooperate to learn anything about a_0.

The client C shares a secret value a_0 by choosing values $a_1, \ldots, a_{k-1}$ uniformly at random from F, and forms the polynomial $p(x) = \sum_{i=0}^{k-1} a_i x^i$. Then, C computes and distributes the secret shares $s_1 = p(1), \ldots, s_N = p(N)$ to the servers $S_1, \ldots, S_N$, respectively.

Addition is easy for the servers to compute, since they can simply add their shares of two values pointwise: if the values s_i form a sharing of a_0 via p, and t_i form a sharing of b_0 via q, then $s_i + t_i$ form a sharing of $a_0 + b_0$ via $p + q$. Similarly, if the values s_i form a sharing of a_0 via p, then, for a constant c, $c \cdot s_i$ form a sharing of $c \cdot a_0$ via $c \cdot p$. Multiplication of two secret values is more complicated, because multiplication of polynomials increases their degree. The solution involves computing and communicating a new sharing, which increases the cost because the servers must communicate.

3 Language design for Secure Cloud Computing (SCC)

For the purpose of analysis, we present a functional language whose definition is parameterized by a set of given operations over some given type of encryptable values. This language, $\lambda^{\rightarrow}_{\mathrm{P,S}}$, is a form of simply-typed lambda calculus, with labeled types as used in information flow languages (see, e.g., [26]). Our implementation, described in Section 4, embeds an extension of this language in Haskell, and provides specific operations over encryptable integer values. From the programmer's standpoint, different cryptographic backends that support the same operations provide the same programming experience. However, our analysis of security and correctness depends on the number of servers, the form of cryptography used, and the form and extent of communication between servers.

In order to provide a uniform analysis encompassing a range of cryptographic alternatives, we formulate both a standard reference semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$ and a distributed semantics that allows an arbitrary number of servers to communicate with the client and with each other in order to complete a computation. Correctness of each distributed cryptographic semantics is proved by showing an equivalence with the reference semantics. Security properties are proved by analyzing the information available to each server at every point in the program execution.

Before presenting the definition of $\lambda^{\rightarrow}_{\mathrm{P,S}}$, we summarize the semantic structure used in our analysis. As shown below, our semantic structure is sufficient to prove correctness and security theorems for $\lambda^{\rightarrow}_{\mathrm{P,S}}$, and general enough to encompass secret sharing, homomorphic encryption, and other platforms.

Reference semantics primitives In the reference semantics, the private values used in computation are interpreted using a set Y of base values, together with primitive operations $\mathrm{op}_1, \ldots, \mathrm{op}_r : Y \times Y \to Y$. For simplicity, we consider only binary operators over a single set of base values. The generalization to arbitrary typed operations over several types of base values is straightforward. We note that this parameterization allows our language (and its Haskell implementation) to easily encompass a variety of platforms, including cryptosystems that are only additively or multiplicatively (rather than fully) homomorphic.

Randomness Because cryptographic primitives used by each of N servers in the distributed semantics may require randomness, we assume a set $\mathcal{R}$ of tuples of sequences, where each $R = (R_C, R_{S_1}, \ldots, R_{S_N}) \in \mathcal{R}$ provides $N+1$ infinite sequences of elements of some finite set Z (such as $Z = \{0,1\}$). As the notation suggests, if there are $N+1$ parties, the sequence R_P is assumed available to the party $P \in \{C, S_1, \ldots, S_N\}$. Since security relies on correct random sequences, we let $U_{\mathcal{R}}$ be a *uniform randomness source:* $U_{\mathcal{R}} = ((U_{\mathcal{R}})_C, (U_{\mathcal{R}})_{S_1}, \ldots, (U_{\mathcal{R}})_{S_N}) = (U_Z^\omega, U_Z^\omega, \ldots, U_Z^\omega)$, where U_Z^ω denotes an infinite sequence of uniform random variables over Z.

Distributed computing infrastructure We assume N servers, $S_1, \ldots, S_N$, execute the secure computation on behalf of one client, C; the extension to multiple clients is straightforward. (In many natural cases, such as homomorphic encryption, $N = 1$). The $(N+1)$ parties will communicate by sending messages via secure party-to-party channels; we denote by M the set of possible message values that may be sent. A *communication round* is a set $\{(P_1^{(i)}, P_2^{(i)}, m^{(i)})\}_{1 \le i \le r}$ of triples, each indicating a sending party, a receiving party, and a message $m \in M$. A *communication trace* is a sequence of communication rounds, possibly empty, and $\mathcal{T}$ is the set of communication traces.

If $A \subseteq \{S_1, \ldots, S_N\}$ is any subset of the servers, the *projection of trace* T *onto* A, written $\Pi_A(T)$, is the portion of the trace visible to the servers in A, i.e., $\Pi_A(\varepsilon) = \varepsilon$ and:

$$\Pi_A(\{(S_1^{(i)}, S_2^{(i)}, m^{(i)})\} \| T) = \{(S_1^{(i)}, S_2^{(i)}, m^{(i)}) \mid \{S_1^{(i)}, S_2^{(i)}\} \cap A \neq \emptyset\} \| \Pi_A(T)\,.$$

General form of cryptographic primitives We work with a two-element security lattice, $\mathscr{S} = \{\mathrm{P}, \mathrm{S}\}$ (with $\mathrm{P} \sqsubseteq \mathrm{S}$), representing (respectively) "public" values, which are transmitted in the clear and may be revealed to any party; and "secret" values, which are encrypted or otherwise hidden, and must remain completely unknown to the adversary. We assume a set $\mathscr{E}_{\mathrm{S}}(Y)$, holding "secret equivalents" of base values in Y; for notational uniformity, we also define $\mathscr{E}_{\mathrm{P}}(Y) = Y$, signifying that the "public equivalent" of a value is just the value itself.

We also assume a cryptographic protocol operation $\mathrm{Init} : \mathcal{R} \to \mathcal{I} \times \mathcal{R}$ that establishes the initial parameters of the platform (e.g., it may generate a public/private key pair for use throughout the computation). Init is a randomized operation, taking a random source in $\mathcal{R}$ (and returning the modified source after potentially consuming some values). We define $\hat{\iota}$ to be the random variable over $\mathcal{R}$ that is derived from running Init on a uniformly random source ($\hat{\iota} = \pi_1(\mathrm{Init}(U_{\mathcal{R}}))$). Further, we assume a projection operator from the initial parameters onto any collection of servers $A \subset \{S_1, \ldots, S_n\}$, writing $\Pi_A(\iota)$ to mean, intuitively, the portion of the initial parameters ι that servers in A should receive.

The other cryptographic operations used in the distributed semantics return secret or public values, but may also consume random values ($\mathcal{R}$), read from the initial parameters ($\mathcal{I}$), and/or result in communication among the parties ($\mathcal{T}$). We assume the following operations:

- $\mathrm{Enc}_{\mathrm{S}} : Y \times \mathcal{R} \times \mathcal{I} \to \mathscr{E}_{\mathrm{S}}(Y) \times \mathcal{R} \times \mathcal{T}$, "hiding" $y \in Y$.
- $\mathrm{Dec}_{\mathrm{S}} : \mathscr{E}_{\mathrm{S}}(Y) \times \mathcal{R} \times \mathcal{I} \to Y \times \mathcal{R} \times \mathcal{T}$, "unhiding".
- $\mathrm{Enc}_{\alpha,\beta}(\mathrm{op}_i) : \mathscr{E}_{\alpha}(Y) \times \mathscr{E}_{\beta}(Y) \times \mathcal{R} \times \mathcal{I} \to \mathscr{E}_{\alpha \sqcup \beta}(Y) \times \mathcal{R} \times \mathcal{T}$ (when $\alpha \sqcup \beta = \mathrm{S}$), evaluating a primitive.

For notational uniformity, as above, we also define the corresponding operations in the degenerate case of "hiding" public values: $\mathrm{Enc}_{\mathrm{P}}(y, R, \iota) = (y, R, \varepsilon)$, $\mathrm{Dec}_{\mathrm{P}}(y, R, \iota) = (y, R, \varepsilon)$, and $\mathrm{Enc}_{\mathrm{P},\mathrm{P}}(\mathrm{op}_i)(y_1, y_2, R, \iota) = (\mathrm{op}_i(y_1, y_2), R, \varepsilon)$.

In reasoning about the distributed semantics, we require that all of the protocol operations consume randomness sources correctly, i.e., when given random sources $R = (R_C, R_{S_1}, \ldots, R_{S_N})$, each operation returns a tuple $R' = (R'_C, R'_{S_1}, \ldots, R'_{S_N})$, where each R'_P is a suffix of R_P and the entire result of the operation depends only on the prefix consumed (and thus independent of R'_P). As a corollary, any operation given uniform randomness $U_{\mathcal{R}}$ must return $U_{\mathcal{R}}$.

Cryptographic functional correctness We assume the usual encryption and homomorphism conditions, augmented for cryptographic primitives that depend on explicit randomness and that may communicate among servers to produce their result. More precisely, for every $y \in Y$, and every choice of initial parameters $\iota \in \mathcal{I}$, we assume a family of *safe sets* $\mathcal{E}_{\alpha}(y, \iota)$: intuitively, any value $l \in \mathcal{E}_{\alpha}(y, \iota)$ can safely serve as the "hiding" of y under the initial parameters ι (at secrecy level $\alpha \in \{\mathrm{P}, \mathrm{S}\}$). More precisely:

- $\pi_1(\mathrm{Enc}_{\alpha}(y, R, \iota)) \in \mathcal{E}_{\alpha}(y, \iota)$

We also require that unhiding ("decryption") is the left-inverse of hiding ("encryption"), and hiding commutes homomorphically with the primitive operations:

- $\pi_1(\mathrm{Dec}_{\alpha}(\pi_1(\mathrm{Enc}_{\alpha}(y, R_1, \iota)), R_2, \iota)) = y$
- $\pi_1(\mathrm{Enc}_{\alpha,\beta}(\mathrm{op}_i)(l_1, l_2), R_3, \iota) \in \mathcal{E}_{\alpha \sqcup \beta}(\mathrm{op}_i(y_1, y_2))$ whenever $l_1 \in \mathcal{E}_{\alpha}(y_1, \iota)$ and $l_2 \in \mathcal{E}_{\beta}(y_2, \iota)$

Cryptographic statistical correctness Analogous to functional correctness, for every $y \in Y$, and every choice of initial parameters $\iota \in \mathcal{I}$, we assume a family of *safe distributions* $\hat{\mathcal{E}}_\alpha(y, \iota)$ over the safe sets $\mathcal{E}_\alpha(y, \iota)$: intuitively, any distribution $l \in \hat{\mathcal{E}}_\alpha(y, \iota)$ can safely serve as the "hiding" of y under the initial parameters ι (at secrecy level $\alpha \in \{\mathrm{P}, \mathrm{S}\}$), assuming randomness is uniform at all stages. We require that "hiding" a base value using a uniform randomness source must yield a safe distribution:

- $\pi_1(\mathrm{Enc}_\alpha(y, U_\mathcal{R}, \iota)) \in \hat{\mathcal{E}}_\alpha(y, \iota)$

In addition, for any two base values y_1 and y_2, we require that evaluating a primitive operation op_i on safe distributions of these two values must yield a safe distribution of $\mathrm{op}_i(y_1, y_2)$:

- $\pi_1(\mathrm{Enc}_{\alpha,\beta}(\mathrm{op}_i)(l_1, l_2, U_\mathcal{R}, \iota)) \in \hat{\mathcal{E}}_{\alpha \sqcup \beta}(\mathrm{op}_i(y_1, y_2), \iota)$ whenever $l_1 \in \hat{\mathcal{E}}_\alpha(y, \iota)$ and $l_2 \in \hat{\mathcal{E}}_\beta(y, \iota)$

Indistinguishability conditions The distributed threat model may generally involve any set of possible combinations of colluding servers. We formalize this by assuming a family $\mathcal{A} \subseteq 2^{\{S_1, \ldots, S_N\}}$ of sets that we refer to as valid sets of untrusted servers. Intuitively, for any $A \in \mathcal{A}$, we assume the cryptographic primitives are intended to provide security even if an adversary has access to all information possessed by servers in A.

Since different platforms may provide different security guarantees of their primitives, we assume a generic notion of indistinguishability; for the purposes of our examples, we will restrict our attention to information-theoretic indistinguishability and computational indistinguishability (with respect to some security parameter of the implementation), but our results easily generalize. Using the form of indistinguishability provided by the platform in question, we assume that any two sequences of partial traces are indistinguishable if each pair of corresponding partial traces describes either a primitive operation or a "hiding" operation on two safely-distributed values.[1] More precisely, for all $T(\iota) = (T_1(\iota), \ldots, T_r(\iota))$ and $T'(\iota) = (T'_1(\iota), \ldots, T'_r(\iota))$, if for each i, either:

$$\begin{aligned} T_i(\iota) &= \pi_3(\mathrm{Enc}_\mathrm{S}(y_i, U_\mathcal{R}, \iota)) \quad \text{or} \\ T_i(\iota) &= \pi_3(\mathrm{Enc}_{\alpha,\beta}(\mathrm{op}_i)(L_{i,1}(\iota), L_{i,2}(\iota), U_\mathcal{R}, \iota)) \\ &\quad \text{where } L_{i,1}(\iota) \in \hat{\mathcal{E}}_\alpha(y_{i,1}, \iota) \text{ and } L_{i,2}(\iota) \in \hat{\mathcal{E}}_\beta(y_{i,2}, \iota) \\ O(\iota) &= (\pi_A(\iota), \pi_A(T_1(\iota)), \ldots, \pi_A(T_k(\iota))) \end{aligned}$$

(and analogously for O', T', substituting $y'_i, y'_{i,1}, y'_{i,2}$ for $y_i, y_{i,1}, y_{i,2}$), then the distributions $O(\hat{\iota})$ and $O'(\hat{\iota})$ are indistinguishable.

▶ **Definition 1.** We say that a platform $(Z, N, M, \mathscr{E}, \mathrm{Enc}, \mathcal{A})$ is a *secure execution platform* for $(Y, (\mathrm{op}_i))$ if it satisfies all of the assumptions of this section.

3.1 Framework

We introduce a simple language, $\lambda^{\rightarrow}_{\mathrm{P,S}}$, based on the simply-typed lambda calculus with base values and primitive operations. In addition to standard constructs, expressions in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ may include variables bound at the program level by the `read` construct, representing secret values input by the clients before the body of the program is evaluated; these input variables are represented by capital letters X (in contrast to lambda-bound variables, which use lowercase letters x), to emphasize the phase distinction between input processing and evaluation

[1] These values may be either secret (S) or public (P). In the latter case, we still assume that the communication traces are indistinguishable, since a properly implemented protocol should not need to exchange publicly-known information between servers at each operation.

of the program body. Programs in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ may also include `reveal` operations, which specify that the value in question need not be kept secret during the computation. Throughout this section, we assume a set Y, primitive operations (op_i), and a secure execution platform for $(Y, (\mathrm{op}_i))$, as specified in Section 3.

Listing 1 Syntax for expressions and programs.

$$e \;::=\; x \mid \lambda x.e \mid e_1\, e_2 \mid \mathrm{op}_i(e_1, e_2) \mid y \in Y \mid X \mid \texttt{reveal}\ e$$
$$p \;::=\; \texttt{read}\ X_1, \ldots, X_r\,; e$$

The static semantics (Listing 2) are standard; we assume the two-element security lattice $\{\mathrm{P}, \mathrm{S}\}$, $\mathrm{P} \sqsubseteq \mathrm{S}$, denoting the types of (respectively) public values, which may be revealed to any party (including the servers); and secret values, about which the protocol may reveal no information. Note that we include both the static semantics for expressions $(\Gamma \vdash e : \tau)$ and those for values $(\Gamma \vdash_v v : \tau)$.

Listing 2 Static semantics for expressions and values.

$$\frac{}{\Gamma \vdash y : (Y, \mathrm{P})} \qquad \frac{}{\Gamma \vdash X : (Y, \mathrm{S})} \qquad \frac{\Gamma \vdash e : (Y, \mathrm{S})}{\Gamma \vdash \texttt{reveal}\ e : (Y, \mathrm{P})} \qquad \frac{\Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}$$

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\, e_2 : \tau_2} \qquad \frac{\Gamma \vdash e_1 : (Y, \alpha) \qquad \Gamma \vdash e_2 : (Y, \beta)}{\Gamma \vdash \mathrm{op}_i(e_1, e_2) : (Y, \alpha \sqcup \beta)}$$

$$\frac{y \in Y}{\Gamma \vdash_v (y, \alpha) : (Y, \alpha)} \qquad \frac{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}{\Gamma \vdash_v \lambda x.e : \tau_1 \to \tau_2}$$

We give a standard dynamic semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$ (Listing 3), based on the usual evaluation rules for lambda calculus with primitive operations; to simplify notation, we overload the symbol $\downarrow$ to represent the evaluation judgments for programs, $(\kappa, p) \downarrow (v, \mathcal{O})$, as well as those for expressions, $(\kappa, \rho, e) \downarrow (v, \mathcal{O})$. The environment κ represents the initial (secret) values supplied by the client. Operationally, the `read` construct is a no-op, but for clarity we retain it in the syntax, since in the implementation semantics (Listing 5) it will represent the "hiding" and initial transmission of the values from the client to the servers. The `reveal` construct acts as a cast from S to P, and may therefore have side effects in an implementation of $\lambda^{\rightarrow}_{\mathrm{P,S}}$ (as discussed below), but these effects are guaranteed to be benign with respect to functional correctness, since they do not change the first component of the resulting value in the dynamic semantics (Listing 3). We also track a list of "observations", $\mathcal{O}$, throughout the evaluation, holding all values ever supplied to `reveal`; this is important in proving security properties (Theorem 6), as we will show that an appropriately constrained adversary learns nothing except what is entailed by these observations.

We have the usual type safety theorem (encompassing both progress and preservation):[2]

▶ **Theorem 2** (Soundness for Reference Semantics). *If $\emptyset \vdash e : \tau$, $p = \texttt{read}\ X_1, \ldots, X_r; e$, $\mathrm{FV}(e) \subseteq \{X_1, \ldots, X_r\}$, and κ maps each X_r to an element of Y, then there exists a value v and an observation sequence $\mathcal{O}$ such that $(\kappa, p) \downarrow (v, \mathcal{O})$ and $\emptyset \vdash_v v : \tau$.*

[2] For space reasons, we omit the proofs of theorems in this section.

Listing 3 "Reference" dynamic semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$.

$$\frac{}{(\kappa,\rho,y)\downarrow((y,\mathrm{P}),\varepsilon)} \quad \frac{}{(\kappa,\rho,X)\downarrow((\kappa(X),\mathrm{S}),\varepsilon)} \quad \frac{}{(\kappa,\rho,x)\downarrow(\rho(x),\varepsilon)} \quad \frac{}{(\kappa,\rho,\lambda x.e)\downarrow(\lambda x.e,\varepsilon)}$$

$$\frac{(\kappa,\rho,e)\downarrow((y,\mathrm{S}),\mathcal{O})}{(\kappa,\rho,\texttt{reveal}\ e)\downarrow((y,\mathrm{P}),\mathcal{O}\|y)} \qquad \frac{(\kappa,\rho,e_1)\downarrow(\lambda x.e,\mathcal{O}_1) \qquad (\kappa,\rho,e_2)\downarrow(v_2,\mathcal{O}_2) \qquad (\kappa,\rho[x\mapsto v_2],e)\downarrow(v,\mathcal{O}_3)}{(\kappa,\rho,e_1\,e_2)\downarrow(v,\mathcal{O}_1\|\mathcal{O}_2\|\mathcal{O}_3)}$$

$$\frac{(\kappa,\rho,e_1)\downarrow((y_1,\alpha),\mathcal{O}_1) \qquad (\kappa,\rho,e_2)\downarrow((y_2,\beta),\mathcal{O}_2)}{(\kappa,\rho,\mathrm{op}_i(e_1,e_2))\downarrow((\mathrm{op}_i(y_1,y_2),\alpha\sqcup\beta),\mathcal{O}_1\|\mathcal{O}_2)} \qquad \frac{(\kappa,\emptyset,e)\downarrow(v,\mathcal{O})}{(\kappa,\texttt{read}\ X_1,\ldots,X_r;e)\downarrow(v,\mathcal{O})}$$

In order to address correctness and security of implementations, we augment the language $\lambda^{\rightarrow}_{\mathrm{P,S}}$ so that there is an additional case for result values, $l\in\mathscr{E}_{\mathrm{S}}(Y,\mathcal{I})$, representing hidden values; we denote this augmented language by $\hat{\lambda}^{\rightarrow}_{\mathrm{P,S}}$. We give a dynamic semantics for $\hat{\lambda}^{\rightarrow}_{\mathrm{P,S}}$ in Listing 5. In contrast to the first, "reference", dynamic semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$, the "distributed" semantics for $\hat{\lambda}^{\rightarrow}_{\mathrm{P,S}}$ reflects the steps taken by an actual implementation. We again have the usual type safety theorem for $\hat{\lambda}^{\rightarrow}_{\mathrm{P,S}}$ under the distributed semantics:

▶ **Theorem 3** (Soundness for Distributed Semantics). *If $\emptyset\vdash e:\tau$, $p=\texttt{read}(X_1,\ldots,X_r);e$, $\mathrm{FV}(e)\subseteq\{X_1,\ldots,X_r\}$, and κ maps each X_i to an element of Y, then for all $\iota\in\mathcal{I}$, randomness sources $R\in\mathcal{R}$, there exists a value w, a trace T, and a randomness source $R'\in\mathcal{R}$ such that $(\kappa,R,p)\Downarrow(w,R',T)$ and $\emptyset\vdash_{tv} w:\tau$.*

Listing 4 Static semantics for values ("distributed" semantics).

$$\frac{y\in Y \qquad l\in\mathscr{E}_\alpha(y)}{\Gamma\vdash_{tv}(l,\alpha):(Y,\alpha)} \qquad \frac{\Gamma\vdash\lambda x.e:\tau_1\to\tau_2}{\Gamma\vdash_{tv}\lambda x.e:\tau_1\to\tau_2}$$

The reference semantics expresses the standard meaning of programs in $\lambda^{\rightarrow}_{\mathrm{P,S}}$, while the distributed semantics expresses in more detail how an implementation should realize them. Evidently, in a correct system we would expect evaluation to arrive at equivalent results in both cases; this is guaranteed by the following theorem (where the relevant similarity relation is defined in Listing 6):

▶ **Theorem 4** (Functional Correctness). *If $\emptyset\vdash e:\tau$, $p=\texttt{read}(X_1,\ldots,X_r);e$, $\mathrm{FV}(e)\subseteq\{X_1,\ldots,X_r\}$, κ maps each X_i to an element of Y, and $(\kappa,p)\downarrow(v,\mathcal{O})$, then for all $R\in\mathcal{R}$, there exist $R'\in\mathcal{R}$, T, w, and ι such that $(\kappa,R,p)\Downarrow(T,R',w,\mathcal{O})$ and $v\sim^{\emptyset,\iota}_{\tau} w$.*

Functional correctness expresses that for any well-formed randomness source $R\in\mathcal{R}$, regardless of whether it was in fact generated randomly, the distributed semantics yields the correct answer. It will also be useful to have a correctness theorem expressing the behavior of the system when given a truly random source. In particular, if we regard the values in question as random variables, and assume that at the beginning of the computation they satisfy appropriate safe distributions as given by $\hat{\mathcal{E}}_{\{\mathrm{P,S}\}}(\cdot,\cdot)$, we can show that values remain in such distributions throughout the computation (Theorem 5). In order to state this result, we introduce a similarity relation $\approx^{\Gamma,\iota}_{\tau}$ (Listing 6) to relate values in the reference semantics with their safe distributions.

Listing 5 "Distributed" dynamic semantics for $\hat{\lambda}^{\rightarrow}_{\mathrm{P,S}}$.

$$\frac{}{(\iota, \Psi, \Delta, R, y) \Downarrow (\varepsilon, R, (y, \mathrm{P}), \varepsilon)} \qquad \frac{}{(\iota, \Psi, \Delta, R, X) \Downarrow (\varepsilon, R, (\Psi(X), \mathrm{S}), \varepsilon)}$$

$$\frac{}{(\iota, \Psi, \Delta, R, x) \Downarrow (\varepsilon, R, \Delta(x), \varepsilon)} \qquad \frac{}{(\iota, \Psi, \Delta, R, \lambda x.e) \Downarrow (\varepsilon, R, \lambda x.e, \varepsilon)}$$

$$\frac{(\iota, \Psi, \Delta, R, e) \Downarrow (T_1, R_1, (l, \mathrm{S}), \mathcal{O}_1) \qquad (y, R_2, T_2) = \mathrm{Dec}_{\mathrm{S}}(l, R_1, \iota)}{(\iota, \Psi, \Delta, R, \texttt{reveal}\ e) \Downarrow (T_1 \| T_2, R_2, (y, \mathrm{P}), \mathcal{O}_1 \| y)}$$

$$\frac{\begin{array}{c}(\iota, \Psi, \Delta, R, e_1) \Downarrow (T_1, R_1, \lambda x.e, \mathcal{O}_1) \\ (\iota, \Psi, \Delta, R_1, e_2) \Downarrow (T_2, R_2, v_2, \mathcal{O}_2) \qquad (\iota, \Psi, \Delta[x \mapsto v_2], R_2, e) \Downarrow (T_3, R_3, v, \mathcal{O}_3)\end{array}}{(\iota, \Psi, \Delta, R, e_1 e_2) \Downarrow (T_1 \| T_2 \| T_3, R_3, v, \mathcal{O}_1 \| \mathcal{O}_2 \| \mathcal{O}_3)}$$

$$\frac{\begin{array}{c}(\iota, \Psi, \Delta, R, e_1) \Downarrow (T_1, R_1, (l_1, \alpha), \mathcal{O}_1) \\ (\iota, \Psi, \Delta, R_1, e_2) \Downarrow (T_2, R_2, (l_2, \beta), \mathcal{O}_2) \qquad (l, T_3, R_3) = \mathrm{Enc}_{\alpha,\beta}(\mathrm{op}_i)(l_1, l_2, R_2, \iota) \\ T = T_1 \| T_2 \| T_3 \qquad \mathcal{O} = \mathcal{O}_1 \| \mathcal{O}_2\end{array}}{(\iota, \Psi, \Delta, R, \mathrm{op}_i(e_1, e_2)) \Downarrow (T, R_3, (l, \alpha \sqcup \beta), \mathcal{O})}$$

$$\frac{\begin{array}{c}(R_0, \iota) = \mathrm{Init}(R) \qquad \forall i \in \{1, \ldots, N\}.\ T_i = \{(C, S_i, \Pi_{\{S_i\}}(\iota)\} \\ \forall j \in \{1, \ldots, r\}.\ (l_j, R_j, T'_j) = \mathrm{Enc}_{\mathrm{S}}(\kappa(X_j), R_{j-1}, \iota) \\ (\iota, \{X_1 \mapsto l_l, \ldots, X_r \mapsto l_r\}, \emptyset, R_r, e) \Downarrow (T, v, R', \mathcal{O}) \qquad T' = T_1 \| \ldots \| T_N \| T'_1 \| \ldots \| T'_r \| T\end{array}}{(\kappa, R, \texttt{read}(X_1, \ldots, X_r); e) \Downarrow (T', v, R', \mathcal{O})}$$

Listing 6 Similarity relations for functional and statistical correctness.

$$\frac{l \in \mathcal{E}_\alpha(y, \iota)}{(y, \alpha) \sim^{\Gamma,\iota}_{(Y,\alpha)} (l, \alpha)} \qquad \frac{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}{\lambda x.e \sim^{\Gamma,\iota}_{\tau_1 \to \tau_2} \lambda x.e} \qquad \frac{l \in \hat{\mathcal{E}}_\alpha(y, \iota)}{(y, \alpha) \approx^{\Gamma,\iota}_{(Y,\alpha)} (l, \alpha)} \qquad \frac{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}{\lambda x.e \approx^{\Gamma,\iota}_{\tau_1 \to \tau_2} \lambda x.e}$$

▶ **Theorem 5** (Statistical Correctness). *If $\emptyset \vdash e : \tau$, $p = \texttt{read}(X_1, \ldots, X_r); e$, $\mathrm{FV}(e) \subseteq \{X_1, \ldots, X_r\}$, κ maps each X_i to an element of Y, and $(\kappa, p) \downarrow (v, \mathcal{O})$, then there exist T and w such that $(\kappa, U_{\mathcal{R}}, p) \Downarrow (T, U_{\mathcal{R}}, w, \mathcal{O})$ and $v \approx^{\emptyset,\hat{\iota}}_{\tau} w$ (where the semantics judgments are lifted to distributions).*

For security, however, the above results are not sufficient. Rather, we now show that if, during the evaluation of a program in $\lambda^{\rightarrow}_{\mathrm{P,S}}$, an adversary is confined to observing the data visible to a valid subset of untrusted servers $A \in \mathcal{A}$ (represented by their views of the communication trace), then that adversary learns nothing about the initial secret client values that was not already implied by the observations from `reveal`:

▶ **Theorem 6** (Security). *If $\emptyset \vdash e : \tau$, $p = \texttt{read}(X_1, \ldots, X_r); e$, $(\kappa, U_{\mathcal{R}}, p) \Downarrow (T, U_{\mathcal{R}}, v, \mathcal{O})$, and $(\kappa', U_{\mathcal{R}}, p) \Downarrow (T', U_{\mathcal{R}}, v', \mathcal{O})$, then for all valid sets of untrusted servers $A \in \mathcal{A}$, the distributions $\Pi_A(T)$ and $\Pi_A(T')$ are indistinguishable (in the sense specified by the secure execution platform, as described in Section 3).*

We remark that although the conclusion of this theorem seems simple, it requires some care to set up the proof correctly. In particular, we can proceed by showing inductively that the two evaluation derivations take the same form, with all resulting values, observations, and traces being structurally equal; moreover, all traces can be decomposed into secret components (which, by statistical correctness, must satisfy the hypothesis of the indistinguishabil-

ity assumption), and public components (which are identical between T and T', since both evaluations yield the same observations). We may then conclude indistinguishability of the projections $\Pi_A(T)$ and $\Pi_A(T')$.

3.2 Shamir secret sharing

We now define Shamir secret sharing in the notation of our framework (Section 3), and show that it is a secure execution platform (Definition 1) for addition and multiplication over a finite field, thereby concluding all of the correctness and security results of Section 3.1 as applied to $\lambda^{\rightarrow}_{\mathrm{P,S}}$ with these two primitive operations. Let N be the number of servers executing the computation (i.e., we use an (N, k) sharing). The set of base values Y is the finite field $\mathbb{F}_p$, where p is a parameter of the implementation,[3] equipped with the usual operations of addition and multiplication ($\mathrm{op}_1(x, y) = (x + y) \bmod p$, $\mathrm{op}_2(x, y) = (x \cdot y) \bmod p$). The sets M of messages and Z of random numbers are also defined to be $\mathbb{F}_p$. We define the set of "hidden equivalents" $\mathscr{E}_{\mathrm{S}}(\mathbb{F}_p)$ to be $\mathbb{F}_p^N$; during computations, we will be concerned specifically with inhabitants of $\mathscr{E}_{\mathrm{S}}(\mathbb{F}_p)$ that represent each of the N servers' shares of some base value. Apart from the initial secret sharing, there is no initialization phase, so we let $\mathcal{I}$ be the singleton set $\{()\}$.

The "hiding" and "unhiding" operations are defined using the standard Shamir secret sharing constructions, as described in Section 2. (For brevity, we defer the formal definitions to the extended version of this article[4].) The primitive operations $\mathrm{Enc}_{\mathrm{S,S}}(+)$, $\mathrm{Enc}_{\mathrm{S,P}}(+)$, $\mathrm{Enc}_{\mathrm{P,S}}(+)$, $\mathrm{Enc}_{\mathrm{S,S}}(*)$, $\mathrm{Enc}_{\mathrm{S,P}}(*)$, and $\mathrm{Enc}_{\mathrm{P,S}}(*)$ are defined similarly, following Section 2; We note that each of the secret sharing operations consumes randomness correctly, by definition. Further, since any base value has only one distribution that can result from using uniform randomness (namely, the uniform distribution over all valid sharings), we define the set of safe distributions to contain only this one: $\hat{\mathcal{E}}_{\mathrm{S}}(n, ()) = \{\mathcal{D}(n)\} = \{\mathrm{Enc}_{\mathrm{S}}(n, U_{\mathcal{R}}, ())\}$. We also define $\mathcal{A}$, the family of valid untrusted subsets of the servers, to include exactly those subsets with cardinality less than k, and we specify that the system should provide information-theoretic security. Assuming this specification, the required functional correctness, statistical correctness, and indistinguishability properties of the primitives follow from the properties of secret sharing outlined in Section 2. (Again for brevity, we omit proofs of all of these properties, but we refer the reader to the extended version.)

Given that the operations of Shamir secret sharing satisfy all of the required properties (as enumerated in Section 3), we can conclude that Shamir secret sharing is a secure execution platform for $(+, \times)$, and thus all of the results of Section 3.1 hold of programs in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ when it is given the semantics of Shamir secret sharing. In particular, functional correctness (Theorem 4) takes on the flavor of a "SIMD" property, stating that the evaluation of a program on N servers results in N-tuples in the "distributed" semantics (a share for each server) being produced in lock-step with their equivalents (the shared value) in the "reference" semantics. Moreover, the security result (Theorem 6) now guarantees the desired secrecy property for the entire language: if the adversary can observe the data from at most k of the servers, then even with unbounded computational resources, it cannot distinguish between any two initial secret value environments, except to the extent that they cause different values to be provided to explicit "`reveal`" directives in the program.

---

[3] In practice, it is more useful to have programs act on integers rather than elements of a finite field. This can be done via a static analysis that infers the largest possible integer value that can arise during the execution, given bounds on the input values.

[4] Available at `http://eprint.iacr.org/2011/561`.

3.3 Fully homomorphic encryption

In addition to secure multiparty computation, a variety of homomorphic encryption schemes can also serve as secure execution platforms for standard primitive operations. In particular, we will now show that any fully homomorphic encryption scheme, and notably Gentry's scheme [11] (under the appropriate cryptographic assumptions), is a secure execution platform for addition and multiplication over the ring $\mathbb{Z}_{2^k}$, achieving security against a computationally-bounded adversary.

In fully homomorphic encryption, the number of servers, N, is 1; the client simply sends encrypted values to the server, and the server performs the computation homomorphically, returning the encrypted result. Although traditionally the operations provided under fully homomorphic encryption would be a complete set of circuit gates, in order to provide a better analogy with secret sharing we define the set of base values Y to be the ring $\mathbb{Z}_{2^k}$, and the operations $(\mathrm{op}_1, \mathrm{op}_2)$ to be addition and multiplication in the ring. The initialization step is just $\mathrm{Init} = \mathsf{KeyGen}(\lambda)$ generating the public/private key pair,[5] where λ is the security parameter to the system.

To begin the computation, the client sends the public key to the server (i.e., $\Pi_{\{S_1\}}(\iota)$ here is $\Pi_{\{S_1\}}((\mathrm{sk}, \mathrm{pk})) = \mathrm{pk}$), then encrypts all of the initial values one bit at a time and sends the corresponding ciphertexts to the server (i.e., $\mathrm{Enc}_{\mathrm{S}}(b_k b_{k-1} \cdots b_1, (\mathrm{sk}, \mathrm{pk})) = (\Psi, \{(C, S_1, \Psi)\})$ where $\Psi = (\mathsf{Enc}(\mathrm{pk}, b_1), \ldots, \mathsf{Enc}(\mathrm{pk}, b_k)))$. During the computation, the server itself performs additions and multiplications on the ciphertexts by homomorphically evaluating the corresponding circuits, producing no communication trace with the client (i.e., $\mathrm{Enc}_{\mathrm{S,S}}(\mathrm{op})(\Psi_1, \Psi_2, (\mathrm{sk}, \mathrm{pk})) = (\mathsf{Eval}(\mathrm{pk}, \mathrm{op}, \Psi_1, \Psi_2), \varepsilon)$); when one of the operands is a public value (i.e., $\mathrm{Enc}_{\mathrm{S,P}}$, $\mathrm{Enc}_{\mathrm{P,S}}$), the server simply "hides" it using $\mathsf{Enc}(\mathrm{pk}, \cdot)$, and then uses $\mathrm{Enc}_{\mathrm{S,S}}$. For `reveal` operations, the server sends back to the client a tuple of ciphertexts to be decrypted, and the corresponding plaintexts (bits of some base value) are returned to the server (i.e., $\mathrm{Dec}_{\mathrm{S}}(\Phi, (\mathrm{sk}, \mathrm{pk})) = (n, \{(S_1, C, \Phi), (C, S_1, n)\})$ where $n = \sum_{i=1}^{k} b_i 2^i$, $b_i = \mathsf{Dec}(\mathrm{sk}, \Phi_i)$). Finally, given these operations, we note that the set $\mathscr{E}_{\mathrm{S}}(Y)$ of possible "hidden" values should be defined as the set of k-tuples of ciphertexts, while the set M of messages in M consists of ciphertexts, plaintexts, and k-tuples of ciphertexts.

Functional correctness of the primitives follows directly from the homomorphic properties of the encryption scheme. For statistical correctness, we can trivially define a safe distribution to be any distribution $l \in \hat{\mathcal{E}}_{\mathrm{S}}(y, \iota)$. Indistinguishability is then immediate for partial traces derived from op_i, since these operations produce empty traces. For the other partial traces (i.e., the initial encryptions $\mathrm{Enc}_{\mathrm{S}}$), indistinguishability follows from CPA-security of the encryption scheme, since the only values in the traces are the encryptions of each of the bits of the secret client inputs.

Thus, fully homomorphic encryption is a secure execution platform for $(+, \times)$, and as above, all of the results of Section 3.1 hold of programs in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ when it is given the semantics of fully homomorphic encryption (now obtaining security guarantees against a computationally bounded adversary).

4 Implementation

We implemented the language of Section 3 as an EDSL in Haskell. Our implementation framework consists of a module that defines the language interface, and SMC and FHE

[5] For clarity, we elide the randomness sources in this section.

libraries that implement the interface combinators. In this section we detail the EDSL and underlying libraries.

4.1 Haskell Secure Cloud Computing EDSL

Our EDSL defines a generic interface, extending the language given in Listing 1. We use the type alias `BType` to denote the base type Y, and `LType` to denote the hidden, or lifted, type $\mathscr{E}_{\mathrm{S}}(Y)$. Additionally, we provide `SIO`, a "secret" `IO` monad, which is used to carry out IO operations and thread platform state (e.g., $\mathcal{R}$ and $\mathcal{T}$ of Section 3) through a given computation.

As previously mentioned, we use Haskell type classes to overload the operators core to the EDSL syntax. As many library functions have side effects (e.g., the SMC multiplication requires network communication) we prefix the EDSL operators with '`.`', and functions with '`s`', as to avoid name collisions with the standard `Prelude` library that is implicitly imported by every Haskell module. Below we detail some of the core aspects of our EDSL. However, we note that, compared to SMCL and other, similar, DSLs, we do not provide any loop constructs—our Haskell embedding allow a programmer to use existing high-order constructs (including general recursion) to create very powerful application-specific loop constructs.

Primitive operations Secure addition, subtraction and multiplication operators are defined using the multi-parameter type class `EDSLArith`. The use of multi-parameter type classes allows us to define instances of the operators with operands of mixed secrecy types (e.g., addition of a public and hidden type). In a similar fashion, we provide standard comparison operators, and a random number generator (RNG) interface. The RNG implementation is, however, limited to SMC following [25].

We leverage Haskell's strong type system (and `newtype` declaration) to provide a hidden Boolean type. Specifically, we introduce `BoolLType` as a wrapper for `LType`, hiding the constructor from the programmer (to avoid unsafe coercions). However, we provides basic Boolean arithmetic and logic operators, including bit–and, bit–or, bit–exclusive-or, $\wedge, \vee$, and $\neg$. Directly, our EDSL can be used to enforce safety of conditionals on hidden values. Specifically, we provide the construct `sif` c `sthen` x `selse` y, which is implemented by safe arithmetization (i.e., $c \cdot x + (1 - c) \cdot y$, that preserves/restores types). In addition to type-safety, this allows writing code using familiar syntax. For example, we can write the max function simply as: `max x y = sif (x .<= y) sthen y selse x`.

Hiding and unhiding functions Further using type classes, we define the `EDSLHide` class which declares `hide`, a Haskell function corresponding to $\mathrm{Enc}_{\mathrm{S}}$; `hide` maps values to their secret equivalent. Dually, we declare `reveal` and the `EDSLReveal` type class that implements the functionality of $\mathrm{Dec}_{\mathrm{S}}$ of Section 3; `reveal` maps hidden, or secret, values to their public equivalent.

User I/O We provide three combinators for interacting with users: `uRead`, `uWrite`, and `uPutStrLn`. `uRead` is used to request a user for input; the user responds by sending a hidden value to the server(s). Dually, `uWrite` is used to send a hidden value to the user, who then locally unhides the value. Observe, that, using

```
withUsers  :: (BType → SIO α) → SIO [α]
withUsers_ :: (BType → SIO α) → SIO ()
```

Figure 4 Iterating over users

this construct, a programmer can write a program that reveals results only to clients. Finally, `uPutStrLn` is used to print a string on a user's terminal. To execute IO actions on all the connected user clients, we provide `withUsers` and `withUsers_`, shown in Figure 4. The former executes a function on all the clients, returning a list of results, while the latter discards the results (useful, e.g., when executing `uWrite`).

4.2 SMC & FHE Library Implementations

In this section we present our SMC and FHE libraries, which instantiate our EDSL with the secure execution platform respectively based on Shamir secret sharing and the Gentry-Halevi FHE implementation [14, 13]. In our framework, each program, such as that of Figure 2, that is executed by the Cloud server parties is actually an `SIO` action. Hence, all the configuration details (e.g., which clients are connected, or the identity of the executing server) are transparent and abstracted into this underlying monad and EDSL constructs. A programmer need only provide an initial configuration that specifies the participating server and client parties, in addition to the program. The same program and configuration is copied to all the Could servers—in the SMC case, the servers execute in a network-SIMD fashion, while in the FHE case the server executes in a standard (network-SISD) fashion. Clients, on the other hand, are event-based: they await instructions from the server(s) and simply respond accordingly. Below, we detail the core base and hidden types, and library-specific details on parties and the execution environment.

Secure Multi-party Computation To implement Shamir secret sharing, we define a base type `Zp` that represents elements of $\mathbb{F}_p$ as a wrapper for the Haskell's arbitrary precision `Integer` type with the standard operators corresponding to their finite field counterparts. Directly, we define a share, or hidden, type (`Share`) as a record enclosing a party number and share value, each of type `Zp`. Shamir secret sharing functions, described in Section 2, are shown in Figure 5, where the type `SMCScheme` is used to encode the (N, k)-scheme. Here, `share` breaks an element into shares, while `reconstruct` takes a list of shares and constructs the corresponding element. We highlight that `share` returns an `SIO` action: the function requires a RNG (we use a cryptographically secure deterministic random bit generator) to break an element into its shares, while `reconstruct` is pure. Further, we highlight, that, compared to the semantics of Listing 5, the RNG in part of underlying monad and not explicitly passed to functions.

```
share :: SMCScheme → Zp → SIO [Share]
reconstruct :: [Share] → Zp
```

Figure 5 Shamir sharing constructs

As previously mentioned, our implementation relies on the notion of party, which we realize using the data type `Party`. A `Party` has an identifier, a network address (hostname, port, SSL certificate), and two typed communication channels: an inbox and outbox. After setting up a mutually-authenticated connection, parties can exchange message using the inbox/outbox channels. Specifically, parties can exchange messages of several forms: (i) a response (constructed with `RespShare`) when sending a server party a share from either a client or another server party, (ii) a request (`ReqShare`) when requesting a client for input, (iii) a reconstruct (`ReconstrShare`) when sending a client a share, who then combines all the received shares to reconstruct the hidden value, (iv) a print (`PrintStr`) when writing a string message to the user's terminal, and (v) a disconnect (`Disconnect`) when the computation has terminated, or failed. We found these message forms to be sufficient when implementing the core Shamir secret sharing EDSL constructs.

Each server party executes an SMC computation in two steps. First, each server listens for incoming connections from other server or client parties. Upon accepting a connection from a party it spawns two threads: a thread that reads incoming network messages and writes them to the inbox channel, and a thread that block-reads the outbox channel, serializes the message, and writes them to the network. Second, when all the servers are interconnected and every client is connected to all the servers, the server parties execute the EDSL program in lock-step, or SIMD fashion. The underlying monad abstracts-away and manages all the configuration details, such as to which party or channel a share should be sent. Of course, the configuration details are queried and used by constructs such as the multiplication operator `(.*)`.

Fully Homomorphic Encryption The Gentry-Halevi C++ implementation [14, 13] provides several functions, including a public/private key pair generation function, encryption/decryption functions, a recrypt (ciphertext refreshing) function and simple single-bit homomorphic arithmetic operators. We extend their implementation with k-bit homomorphic addition, multiplication, comparison and equality testing functions. To integrate the (extended) C++ FHE library into our Haskell framework, we further implemented C wrappers for the basic FHE operations, and various library functions—calling foreign functions in Haskell is accomplished using the Foreign Function Interface (FFI), which is currently best suited for interfacing with C.

Similar to the SMC case, we define a base and hidden type. Specifically, we define the base type (`ZZ`) as a simple wrapper for Haskell's `Int`, bounding it to k-bits. The hidden, encrypted, type is a wrapper for a C pointer (to a vector of "big integers") that allows for simple calling of the C/C++ FHE functions from Haskell. Although this adds the additional complexity of performing garbage collection of the C-allocated big integers, it allows us to use the optimzied C/C++ FHE functions when implementing the EDSL combinators such as the addition operator `(.+)`.

To support a practical Cloud-oriented FHE library, we require the separation of client and server code, and we thus provide functions that serialize and deserialize encrypted values. Directly, this allows for transmission of encrypted values over the network. From a networking perspective the FHE setting is a special case of SMC with $N = 1$. Hence, the FHE notion of party is similar to that of SMC described above, though it additionally requires associating public-private keys with a computation. However, among other differences, compared to SMC, where only server communication is necessary in unhiding, or decrypting, a value, in the FHE setting, communication with a client is necessary. These details are, of course, abstracted into the underlying `SIO` monad and corresponding EDSL constructs (e.g., `reveal`) and thus transparent to the programmer.

4.3 Performance Evaluation

Our SMC library, including the EDSL interface, and comparison protocol of [4], was implemented in roughly 1300 lines of Haskell code. Our FHE library was implemented in about 1200 lines of Haskell, and 650 lines of C/C++ code extending the Gentry-Halevi implementation. To evaluate the performance of these implementation we also implemented various programs, including the Clock-Auction, and mall benchmark suite of [24]. The suite consists of 3 programs that compute the sign of a quadratic polynomial: (i) the *ideal* program operates solely on hidden values, (ii) the *pragmatic* program operates on mixed-secrecy values—all values are secrec except for the evaluation point and the result of the polynomial evaluation, (iii) the *public* program operates solely on public values.

Table 1 Performance benchmarks for SMC and FHE, where the security parameter λ^{toy} corresponds to a "toy" security level (a lattice of with dimension 128). Tests with realistic parameters are currently unfeasible.

Scheme	Ideal	Pragmatic	Public
SMC $(3, 1)$	0.97 sec	3.3 ms	< 1 ms
SMC $(5, 2)$	1.02 sec	3.3 ms	< 1 ms
SMC $(7, 3)$	1.04 sec	3.3 ms	< 1 ms
FHE λ^{toy}	17.6 min	5.3 min	< 1 ms

Table 1 presents our results for various SMC configurations and a "toy" FHE configuration. The SMC implementation uses arithmetic modulo the largest 32-bit prime, while the FHE implementation operates on 8-bit integers. Our experimental setup consisted of 7 machines, interconnected on a local Gig-E network, each machine containing two Intel Xeon E5620 (2.4GHz) processors and 48GB of RAM. Similar to the results of SMCL [24], we observe that the SMC pragmatic version is an order of magnitude faster than the ideal. Compared to their results, our system is significantly faster; however, this is not a meaningful comparison because we are using newer generation of hardware. More importantly, we note that the performance results of both the ideal and pragmatic SMC benchmarks highlight the usability of our SMC implementation for real-world applications.

5 Related work

Among several projects demonstrating potential applications of secure multiparty computations, SCET [5], with its focus on economic applications, implemented secure double auction. In Fairplay [22], programs written in SFDL were converted to primitive operations on bits. Fairplay was restricted to only two parties; this drawback was removed in FairplayMP. Sharemind [3] aimed at general multiparty computation on large datasets, supporting three players and providing security against a passive adversary.

VIFF [8] provides a basic language embedded in Python and API calls to cryptographic primitives. It provides Shamir and pseudorandom secret sharing as options to the programmer. VIFF can be seen as a system for expert programmers to build complex cryptographic protocols. Indeed, VIFF has been used for building distributed implementations of RSA and AES. In contrast, our EDSL is for writing applications by nonexpert programmers, and permits one to write at a substantially higher level of abstraction than that of the cryptographic primitives. Moreover, compared to Python, Haskell has a natural advantage as a host for EDSLs; as a functional language, Haskell allows extensive static reasoning about programs, performs a variety of optimizations, and has lightweight multithreading capabilities. On the other hand, our EDSL can complement systems such as VIFF by targeting it as a platform, providing a higher-level abstraction layer over its powerful and efficient cryptographic primitives.

From a theoretical standpoint, the systems discussed above are generally concerned with implementing cryptographic protocols, without proving the more comprehensive correctness and security properties we consider. The closest work is SMCL [24], an imperative-style DSL. The papers on SMCL contain proofs of correctness and security properties, but they do not formally define a crucial aspect: the requirements on the side-effects produced by primitive operations so that security can be guaranteed. Our system is also implemented

as an EDSL, rather than as a standalone language, so it can leverage the full power of Haskell and its type system. In addition, unlike SMCL, our system easily generalizes to other cryptographic schemes. As far as we know, we are the first to formalize and prove correctness and security properties for a unified language framework which encompasses a wide range of cryptographic schemes for computation on encrypted data, in particular Shamir secret sharing and fully homomorphic encryption.

6 Conclusions

We present the design, foundational analysis, implementation, and performance benchmarks for an embedded domain-specific language that allows programmers to develop code that can be run on different secure execution platforms with different security guarantees. We prove functional correctness and confidentiality for any *secure execution platform* meeting our definitions and then show that a specific secret-sharing scheme and fully homomorphic encryption both meet our definition. Our language allows developers to produce a single program that can be executed on different secure execution platforms, making the deployment decisions after development according to security and performance requirements.

As a programming language, our embedded DSL, implemented as a Haskell library, allows developers to use standard Haskell software development environments. Programmers also have the benefit of sophisticated type-checking and general programming features of Haskell because we rely only on the Haskell type discipline to enforce information flow and other restrictions; there are no unexpected *ad hoc* code restrictions. Our Haskell implementation also provides more flexible data structures than previous work because our information-flow constraints make secrecy-preserving operations on such such structures possible. In future work, we plan to improve the expressiveness of the programming language through more sophisticated information-flow typing of recursive and iterative constructs, for example. In addition, we plan to apply our framework to other secure execution platforms that can provide stronger guarantees, such as security against active adversaries. We will also explore the possibility of proving formally that a particular implementation realizes our secret semantics, possibly in a mechanically-verified fashion. Finally, we plan to develop more sophisticated implementation techniques, possibly leveraging Template Haskell metaprogramming, such as automatically producing code that is optimized for particular forms of partially homomorphic encryption with better performance.

Acknowledgments This work was supported by DARPA PROCEED, under contract #N00014-11-1-0276-P00002, the National Science Foundation, and the Air Force Office of Scientific Research. D. Stefan and J. Zimmerman are further supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

References

1 B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

2 M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

3 D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206, 2008.

4 P. Bogetoft, D. L. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. `http://eprint.iacr.org/`.

5 P. Bogetoft, I. B. Damgård, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft. Secure computing, economy, and trust: A generic solution for secure auctions with real-world applications. Technical Report RS-05-18, BRICS, 2005.

6 D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

7 R. Cramer, I. Damgård, and U. M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

8 I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography*, pages 160–179, 2009.

9 I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010.

10 R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

11 C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

12 C. Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, 2010.

13 C. Gentry and S. Halevi. Gentry-Halevi implementation of a fully-homomorphic encryption scheme. `https://researcher.ibm.com/researcher/files/us-shaih/fhe-code.zip`.

14 C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, pages 129–148, 2011.

15 C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple bgn-type cryptosystem from lwe. In *EUROCRYPT*, pages 506–522, 2010.

16 O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

17 Embedded domain-specific languages in haskell. `http://www.haskell.org/haskellwiki/Research_papers/Domain_specific_languages`.

18 Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

19 Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, pages 577–594, 2010.

20 Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.

21 Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

22 D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

23 M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *STOC*, pages 590–599, 2001.

24 J. D. Nielsen and M. I. Schwartzbach. A domain-specific programming language for secure multiparty computation. In *PLAS*, pages 21–30, 2007.

25 T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Public Key Cryptography*, pages 343–360, 2007.

26 A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 21(1):2003, 2003.

27 M. C. Silaghi. Smc: Secure multiparty computation language. `http://www.cs.fit.edu/~msilaghi/SMC/tutorial.html`, 2004.

28 N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography*, pages 420–443, 2010.

29 M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

30 A. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

Schema Mappings and Data Examples: Deriving Syntax from Semantics

Phokion G. Kolaitis[1]

1 **University of California Santa Cruz & IBM Research - Almaden California, USA**
kolaitis@cs.ucsc.edu

Abstract

Schema mappings are high-level specifications that describe the relationship between two database schemas. Schema mappings are considered to be the essential building blocks in such critical data interoperability tasks as data exchange and data integration. For this reason, they have been the focus of extensive research investigations over the past several years. Since in real-life applications schema mappings can be quite complex, it is important to develop methods and tools for illustrating, explaining, and deriving schema mappings. A promising approach to this effect is to use "good" data examples that illustrate the schema mapping at hand.

In this talk, we present an overview of recent work on characterizing and deriving schema mappings via a finite set of data examples. We show that every LAV schema mapping (i.e., a schema mapping specified by a finite set of local-as-view tuple-generating dependencies) is uniquely characterized by a finite set of universal data examples with respect to the class of all LAV schema mappings. We also show that this type of result does not hold for arbitrary GAV schema mappings (i.e., schema mappings specified by a finite set of global-as-view tuple-generating dependencies). After this, we give a necessary and sufficient algorithmic condition for a GAV schema mapping to be uniquely characterizable by a finite set of universal examples with respect to the class of all GAV schema mappings. Along the way, we establish tight connections between unique characterizability of schema mappings and homomorphism dualities.

This is joint work with Bogdan Alexe (IBM Research - Almaden), Balder ten Cate (UC Santa Cruz), and Wang-Chiew Tan (UC Santa Cruz and IBM Research - Almaden) based on [1, 2, 3].

1998 ACM Subject Classification D.2.12 Interoperability: Data mapping; H.2.5 Heterogeneous Databases: Data translation

Keywords and phrases Schema mappings, database constraints, data exchange, data integration, universal solutions, homomorphism dualities

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.25

References

1 Bogdan Alexe, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 261–272, 2010.

2 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Designing and refining schema mappings via data examples. In *SIGMOD Conference*, pages 133–144, 2011.

3 Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Database constraints and homomorphism dualities. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 475–490, 2010.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 25–25

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Quantum State Description Complexity

Umesh V. Vazirani[1]

1 **University of California, Berkeley,** `vazirani@eecs.berkeley.edu`

Abstract

Quantum states generally require exponential sized classical descriptions, but the long conjectured area law provides hope that a large class of natural quantum states can be described succinctly. Recent progress in formally proving the area law is described.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases area law, Hamiltonian, description complexity, detectability lemma, entanglement

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.26

1 Summary

Arguably the most fundamental difference between quantum and classical systems is the exponential difference in the number of parameters required to describe them - whereas a classical system of n particles can be described by $\mathcal{O}(n)$ parameters, an *arbitrary* state of a similar quantum system would in general require $2^{\Omega n}$ parameters. The reason for this discrepancy is quantum entanglement, which is a fundamental resource in quantum information processing, is also the principal obstacle in the efficient simulation of quantum systems on a classical computer. But is there a significant class of quantum states which can be described succinctly? More formally is there a set S of states such that for every state $\left|\psi\right\rangle \in S$ there is a classical description $w \in \{0,1\}^{\mathrm{poly}(n)}$ that "describes" $\left|\psi\right\rangle$ — in the sense that it is possible to efficiently compute interesting quantities about $\left|\psi\right\rangle$, such as energy or two point correlations, from the classical description w. This may be formalized by saying that the result of any k-local measurement, for some constant k, on $\left|\psi\right\rangle$ can be efficiently computed from w.

A possible candidate for S is the set of ground states of gapped local Hamiltonians. Local Hamiltonians are quantum analogs of CSPs, and ground states are quantum analogs of satisfying assignments or configurations that minimize the number of unsatisfied constraints. More formally, consider a set of n (d dimensional) spins arranged in a lattice, with nearest-neighbor interactions described by a local Hamiltonian $H = \sum_{i=1}^{n-1} H_i$. H is a $d^n \times d^n$ Hermitian matrix, whose eigenvectors are the states of the system with definite energy equal to the corresponding eigenvalue. We assume that H has a unique ground state $\left|\Omega\right\rangle$, the lowest energy eigenvector. The spectral gap of H, denoted by ϵ, is the difference between the two lowest eigenvalues. We will assume that each term H_i has unit norm and say that H is gapped if the spectral gap ϵ is bounded below by some constant independent of n.

A remarkable conjecture in condensed matter physics dating back about a half century is the Area Law, which strongly bounds the entanglement in ground states of gapped local Hamiltonians. More specifically, for any contiguous region of the grid L, the entanglement entropy between the particles inside L and the particles outside L is trivially bounded by ($\log d$ times) the number of particles in L, which we may think of as the volume of L. The surface area of L is the number of grid edges crossing between L and $\overline{L}$ or equivalently

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 26–27

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

thenumber of terms of the Hamiltonian describing interactions between particles in L with particles in $\overline{L}$. Then a state $|\psi\rangle$ obeys an area law if the *entanglement entropy* between L and $\overline{L}$ is upper-bounded by a quantity proportional to the surface area of L, rather than its volume.

A few years ago, in a seminal paper [5] Hastings proved that ground states of gapped 1D systems obey an area law. More specifically, for such systems, he proved that the entanglement entropy across any cut in the chain is bounded by $S_{1D} \leq e^{\mathcal{O}(X)}$, where $X = \frac{\log d}{\epsilon}$. As a consequence he showed that the ground state of such systems has a polynomial classical description by a Matrix Product State of size $ne^{S_{1D}}$.

This left the area law for two (and higher) dimensional systems as one of the most important open questions in Quantum Hamiltonian Complexity. One way to tackle this question is to improve the bound on the entanglement entropy, since a bound of $\mathcal{O}(\log d)$ would automatically imply area laws for higher dimensional systems — simply treat the system as a 1D system by grouping all the particles at the boundary as a huge particle of dimension d^B, where B is the number of particles on the boundary. A logarithmic bound on entanglement entropy now yields a bound of $\mathcal{O}(B \log d))$.

A combinatorial approach to proving the area law for 1D frustration-free systems (i.e., systems where the ground state $|\Omega\rangle$ is also the common ground state of all local terms) was introduced in [2]. The new proof replaced Hastings' analytical machinery, including the Lieb-Robinson bound and spectral Fourier analysis, with the Detectability Lemma [1], a combinatorial lemma about local Hamiltonians. However, the resulting bound was no better than Hastings' bound because, at its heart, the argument followed the same outline as Hastings', including the use of a "monogamy of entanglement"-type argument that leads to an exponential slack.

A completely new approach to proving the 1D area law (based on the detectability lemma) was pursued in [3] and [4]. This led to an exponential improvement in the bound on the entanglement entropy. Formally, in [4] it was proved that for 1D frustration-free systems, the entanglement entropy of every cut in the chain is upper bounded by $S_{1D} \leq \mathcal{O}(X^3 \log^8 X)$. This also improves by an exponential factor the bound on the classical description of such states by a Matrix Product State description. Moreover, by using the locality properties of the Hamiltonian, for the case of 2D systems the entanglement bound can be improved to $S \leq \mathcal{O}(B^2 \cdot X^3 \log^8(B \cdot X))$. This leaves the 2D area law poised at a very interesting point — any non-trivial improvement in the existing bounds would result in the first sub-volume law for 2D systems.

References

1 Dorit Aharonov, Itai Arad, Zeph Landau, and Umesh Vazirani. The detectability lemma and quantum gap amplification. In *STOC '09: Proc. 41st Annual ACM Symposium on Theory of Computing,* `arXiv:0811.3412`, pages 417–426, New York, NY, USA, 2009. ACM.

2 Dorit Aharonov, Itai Arad, Zeph Landau, and Umesh Vazirani. Quantum Hamiltonian complexity and the detectability lemma. *arXiv:1011.3445*, November 2010.

3 Dorit Aharonov, Itai Arad, Zeph Landau, and Umesh Vazirani. The 1d area law and the complexity of quantum states: A combinatorial approach. In *FOCS '11: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, 2011.

4 Itai Arad, Zeph Landau, and Umesh Vazirani. An improved 1d area law for frustration-free systems. 2011. arXiv:1111.2970v1 [quant-ph].

5 MB Hastings. An Area Law for One Dimensional Quantum Systems. *JSTAT, P*, 8024, 2007.

Approximation Algorithms for Union and Intersection Covering Problems

Marek Cygan[1], Fabrizio Grandoni[2], Stefano Leonardi[3], Marcin Mucha[1], Marcin Pilipczuk[1], and Piotr Sankowski[1]

1 **Institute of Informatics, University of Warsaw, Poland.**
{cygan,mucha,malcin,sank}@mimuw.edu.pl
2 **Computer Science Department, University of Rome Tor Vergata, Roma, Italy.**
grandoni@disp.uniroma2.it
3 **Department of Computer and System Science, Sapienza University of Rome, Italy.** stefano.leonardi@dis.uniroma1.it

Abstract

In a classical covering problem, we are given a set of *requests* that we need to satisfy (fully or partially), by buying a subset of *items* at minimum cost. For example, in the k-MST problem we want to find the cheapest tree spanning at least k nodes of an edge-weighted graph. Here, nodes represent requests whereas edges correspond to items.

In this paper, we initiate the study of a new family of *multi-layer* covering problems. Each such problem consists of a collection of h distinct instances of a standard covering problem (*layers*), with the constraint that all layers share the same set of requests. We identify two main subfamilies of these problems:

- in an UNION multi-layer problem, a request is satisfied if it is satisfied *in at least one* layer;
- in an INTERSECTION multi-layer problem, a request is satisfied if it is satisfied *in all* layers.

To see some natural applications, consider both generalizations of k-MST. UNION k-MST can model a problem where we are asked to connect a set of users to at least one of two communication networks, e.g., a wireless and a wired network. On the other hand, INTERSECTION k-MST can formalize the problem of providing both electricity and water to at least k users.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation algorithms, Partial covering problems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.28

1 Introduction

In the fundamental MINIMUM SPANNING TREE problem (MST), the goal is to compute the cheapest tree which spans all the n nodes of a given edge-weighted graph $G = (V, E)$. To handle the subtleties of real-life applications, several natural generalisations and variants of the problem have been considered. For example, in the STEINER TREE problem we need to connect only a given subset W of k *terminal* nodes. In the k-MST problem instead, the goal is to connect at least k (arbitrary) nodes. One common feature of these generalizations is that we need to design a single network. However, this is often not the case in the applications. For example, suppose we want to provide at least k out of n users with both electricity and water. In this case, we cannot design the water and electricity infrastructures independently: our decisions on which users to reach have to be synchronized.

Consider now another classic problem, the TRAVELLING SALESMAN problem (TSP): here we are given a complete weighted graph, and the goal is to compute the minimum-length

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 28–40

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

tour traversing all the nodes. Again, several natural generalizations and variants of the problem have been considered in the literature. Still, all of them deal only with the case where there is a single network. However, there are natural applications which do not fit in this framework. For example, suppose you want to visit a set of places (bank, post office, etc.), and you can use your bike and your car. Of course, you cannot just reach a place by bike, and then suddenly switch to your car (that you left at home). Your trip must consist of a tour by bike and another tour by car, which together touch all the places that you need to visit.

The above examples show the need for a new framework, which is able to capture coordinated decision-making over multiple optimization problems. In this paper we initiate the study of such *multi-layer covering* problems. These problems are characterized by a set of h instances of a standard covering problem (*layers*), sharing a common set of n *requests*. The goal is to *satisfy*, possibly partially, the requests by buying *items* in each layer at minimum total cost. We identify two main families of such problems:

- Intersection problems. Here, as in the water-electricity example, a request is satisfied if it is satisfied *in all* the layers.
- Union problems. Here, as in the car-bike example, a request is satisfied if it is satisfied *in at least* one layer.

1.1 Our results.

We provide hardness and approximation results for the union and intersection versions of several classical covering problems: MST, Steiner Tree, (Nonmetric and Metric) Facility Location, TSP, and Set Cover. (Formal definitions are given at the end of this section). We focus on the partial covering variant of these problems, i.e., k-MST, k-Steiner Tree, etc.: here wet need to satisfy a *target* number k of the n requests. This allows us to handle a wider spectrum of interesting problems. In fact, for intersection problems, if $k = n$ it is sufficient to compute an independent solution for each layer. On the other hand, some of the union problems above are interesting also for the case $k = n$. However, the results that we achieve for that case are qualitatively the same as for $k < n$.

For Intersection versions of k-MST, k-Steiner Tree, k-TSP, k-Set Cover, k-Metric Facility Location, and k-Nonmetric Facility Location, we show that:

- Even for two layers, a polylogarithmic approximation for these problems would imply a polylogarithmic approximation for k-Densest Subgraph. We recall that the best approximation for the latter problem is $O(n^{\frac{1}{4}+\varepsilon})$ [6] and finding a polylogarithmic approximation is a major open problem. Indeed, many researchers believe that a polylogarithmic approximation does not exist, and exploit this assumption in their hardness reductions (see, e.g., [2, 3]).
- On the positive side, we give $\tilde{O}(k^{1-1/h})$-approximation algorithms[1] for these problems.

Note that, in the single-layer case, the above problems can be approximated within a constant or logarithmic factor. Hence, our results show that the complexity of natural intersection problems changes drastically from one to two layers.

For Union versions of k-MST, k-Steiner Tree, k-TSP and k-Metric Facility Location we show that:

- The problems are $\Omega(\log k)$-hard to approximate for an unbounded number h of layers. Furthermore, there is a greedy $O(\log k)$-approximation algorithm. For the first three

[1] The $\tilde{O}$ notation suppresses polylogarithmic factors.

problems this only holds for the rooted version — the unrooted case is inapproximable (i.e., any bounded approximation factor implies $P = NP$).
- There is an LP-based algorithmic framework which provides $O(h)$-approximate solutions. Furthermore, the natural LPs involved have $\Omega(h)$ integrality gap.

We remark that Union k-Set Cover and Union k-Nonmetric Facility Location can be solved by collapsing all layers into one, and hence they are less interesting.

1.2 Related Work.

To the best of our knowledge, and somewhat surprisingly, approximation algorithms for union and intersection problems seem to not have been studied much in the literature. The notable exception is Matroid Intersection, which however is solvable in polynomial time [7]. Some team formation games in social networks can be seen as special cases of our intersection problems [1, 21]. Riaz et al. [25] observe that traditionally, wired and wireless infrastructures have been planned separately, but there is a need for complementary use of wired and wireless technologies in future networks. However, the authors do not consider the optimization aspects of such planning, which is captured by our Union k-MST.

The idea of introducing multiple cost functions into one optimization problem is the main theme of *multi-objective optimization*. Standard and multi-criteria approximation algorithms have been developed for the multi-objective version of several classical problems, such as Shortest Path, Spanning Tree, Matching, etc. (see, e.g., [4, 14, 15, 22, 23]). One could view these problems as having several layers with different costs. However, in contrary to our setting, solutions in different layers have to be *exactly the same*.

Partial covering problems (also known as problems with *outliers*), are well-studied in the literature: e.g., k-MST [12], k-TSP [12], k-Metric Facility Location [19], and k-Set Cover [26]. Their generalization on multiple layers is significantly harder, as our results show. Note that our Union k-Steiner Tree problem generalizes all of the following problems: k-Steiner Tree (and hence k-MST), Prize-Collecting Steiner Tree (see the proof of Theorem 7), and k-Set Cover (see the proof of Theorem 8).

Rent-or-buy and *buy-at-bulk* problems [8, 9, 13, 16] can be seen as multi-layer union problems where edge weights in different layers are related by a multiplicative factor. In contrast, weights of different layers are unrelated in our framework.

Recently Krishnaswamy et al. [20] considered a *matroid median* problem, where the set of open centers must form an independent set in a matroid. We can interpret this as a generalization of a union problem, where we add side constraints between bought items. However, the authors assume that all layers share the same metric space.

1.3 Preliminaries.

In *covering* problems we are given a set $\mathcal{U}$ of n *requests*, and a set $\mathcal{S}$ of *items*, with costs $w : \mathcal{S} \to \mathbb{R}_{\geq 0}$. The goal is to satisfy all requests by selecting a subset of items at minimum cost. We already defined MST, Steiner Tree, and TSP. Here, nodes and edges represent requests and items (with costs $w : E \to \mathbb{R}_{\geq 0}$), respectively. In the Set Cover problem, requests are the elements of a universe $\mathcal{U}$, and items $\mathcal{S}$ are subsets $S_1, \ldots, S_m$ of $\mathcal{U}$. Any S_i satisfies all the $v \in S_i$. Nonmetric Facility Location is a generalization of Set Cover, where we are given a set $\mathcal{F}$ of *facilities*, with opening costs $o : \mathcal{F} \to \mathbb{R}_{\geq 0}$, and a set $\mathcal{C}$ of *clients*, with connection costs $w : \mathcal{C} \times \mathcal{F} \to \mathbb{R}_{\geq 0}$. The goal is to compute a subset $\mathcal{A}$ of *open* facilities such that $\sum_{f \in \mathcal{A}} o(f) + \sum_{c \in \mathcal{C}} w(c, \mathcal{A})$ is minimized. Here, $w(c, \mathcal{A}) := \min_{f \in \mathcal{A}} w(c, f)$. We also say that c is connected to (or served by) $\mathcal{A}(c) := \arg\min_{f \in \mathcal{A}} w(c, f)$. If connection

costs satisfy triangle inequality, the problem is called Metric Facility Location. We can naturally define partial covering versions for the above problems, i.e., k-MST, k-Steiner Tree, k-TSP, k-Nonmetric Facility Location, and k-Metric Facility Location[2].

It is straightforward to define union and intersection versions of the above problems (more details in the corresponding sections). In the rest of this paper, the number of layers is denoted by h, and variables associated to layer i have an apex i (e.g., w^i, o^i, etc.), whereas OPT denotes the optimum solution, and opt its cost. By N we denote the total number of requests and items (in all layers).

By standard reductions, a ρ-approximate algorithm for the k-MST problem implies a 2ρ-approximate algorithm for k-Steiner Tree and k-TSP. Moreover, a ρ-approximation for k-TSP gives a 2ρ-approximation for k-MST. Essentially, the same reductions extend to the union and intersection versions of these problems. For this reason, in the rest of this paper we will consider the union and intersection version of k-MST only.

Proofs and details which are omitted due to lack of space will be given in the full version of the paper.

2 Intersection Problems

In this section we present our main results on the intersection problems. In Intersection k-Set Cover we are given h collections $\mathcal{S}^1, \mathcal{S}^2, \ldots, \mathcal{S}^h$ of subsets of a given universe $\mathcal{U}$, where $w^i : \mathcal{S}^i \to \mathbb{R}_{\geq 0}$ is the cost of subsets in the i'th collection. The goal is to cover at least k elements in all layers simultaneously, at minimum total cost. In Intersection k-MST we are given a complete graph $G = (V, E)$ on n nodes, and h metric edge-weight functions $w^1, \ldots, w^h$. The goal is to compute a tree T^i for each layer such that $\sum_i w^i(T^i)$ is minimized and $|\bigcap_i V(T^i)| \geq k$. In Intersection k-Nonmetric Facility Location we are given a set $\mathcal{F}$ of *facilities*, with opening costs $o^i : \mathcal{F} \to \mathbb{R}_{\geq 0}$ on layer i, and a set $\mathcal{C}$ of *clients*, with connection costs $w^i : \mathcal{C} \times \mathcal{F} \to \mathbb{R}_{\geq 0}$ on layer i. The goal is to compute a subset $\mathcal{A}^i$ of *open* facilities on each layer i and a subset $\mathcal{C}'$ of k clients such that $\sum_i (\sum_{f \in \mathcal{A}^i} o^i(f) + \sum_{c \in \mathcal{C}'} w^i(c, \mathcal{A}^i))$ is minimized. Here $w^i(c, \mathcal{A}^i) := \min_{f \in \mathcal{A}^i}\{w^i(c, f)\}$. Intersection k-Metric Facility Location is the special case of Intersection k-Nonmetric Facility Location where connection costs satisfy triangle inequality.

2.1 Hardness

In order to show the hardness of our problems, we use reductions from the k-Densest Subgraph problem: find the induced subgraph on k nodes with the largest possible number of edges. The fact that partial coverage problems can be as hard as k-Densest Subgraph is already known. Hajiaghayi and Jain [18] use k-Densest Subgraph to show that a partial coverage version of the Steiner Forest problem has no polylogarithmic approximation. In particular they introduce the Minimum ℓ-Edge Coverage problem where one is to find the minimum number of vertices in a graph, whose induced subgraph has at least ℓ edges. Moreover Hajiaghayi and Jain show a relation between approximation ratios for k-Densest Subgraph and Minimum ℓ-Edge Coverage.

[2] In the literature k-Nonmetric Facility Location often means that we are allowed to open at most k facilities, while here we mean that we need to connect at least k clients. Similarly for k-Metric Facility Location. Sometimes k-Set Cover indicates a Set Cover instance where the largest cardinality of a set is k, while our problem is sometimes called Partial Set Cover.

Figure 1 Approximation algorithm for 2-layer Intersection k-Set Cover. For $a \in \{1,2\}$, $\overline{a}$ is the other value in $\{1,2\}$

```
1: procedure SCI(k, U, S^1, S^2, w^1, w^2)
2:     K ← ∅, A^1 ← ∅, A^2 ← ∅
3:     repeat
4:         for a=1 to 2 do
5:             for all X ∈ S^a do
6:                 for b := 1 to min(k − |K|, |X \ K|) do
7:                     Solve one-layer Intersection k-Set Cover problem on layer ā
8:                     with universe X \ K and target b.
9:         Let (a', b', X') be the loop iterators which provide a solution (K', A')
10:        minimizing the ratio of cost C' to number b' of covered elements.
11:        K ← K ∪ K', A^{a'} ← A^{a'} ∪ {X'}, A^{ā'} ← A^{ā'} ∪ A'
12:    until |K| = k
13:    return (K, A^1, A^2)
```

In order to simplify our reductions we extend the result on Minimum ℓ-Edge Coverage to bipartite graphs. In particular, we are able to show that a $f(n)$-approximation for Minimum ℓ-Edge Coverage on bipartite graphs implies a $16(f(2n))^2$-approximation for k-Densest Subgraph on arbitrary graphs. One can naturally reduce a Minimum ℓ-Edge Coverage instance (G,ℓ) on a bipartite graph $G = (V_1 \cup V_2, e)$ to Intersection k-Set Cover (with $k = \ell$), by mapping edges into elements and each node v into the corresponding subset of incident edges $\delta(v) \subseteq E$. A similar reduction works for Intersection k-Metric Facility Location, where facility opening costs are set to 1 and distances corresponding to edges and anti-edges are set to 0 and $+\infty$, respectively. For Intersection k-MST we construct one layer (symmetrically for the other one) by connecting a dummy root node r with all nodes in V_1 at cost 1, and each $v \in V_1$ with $\delta(v)$ at cost 0. As a consequence, we obtain the following two theorems, which suggest that the existence of a polylogarithmic approximation for the considered problems is rather unlikely (or at least hard to achieve).

▶ **Theorem 1.** *If there exists an $f(n)$-approximation algorithm for unweighted* Intersection k-Set Cover *on two layers or for* Intersection k-Metric Facility Location *on two layers, then there exists a $16(f(2m))^2$-approximation algorithm for k-*Densest Subgraph.

▶ **Theorem 2.** *If there exists an $f(n)$-approximation algorithm for* Intersection k-MST *on two layers, then there exists a $16(f(2n+2m+2))^2$-approximation algorithm for k-*Densest Subgraph.

2.2 Intersection k-Set Cover

The basic idea behind our Intersection k-Set Cover algorithm is as follows. We consider any set X in any layer, and any number $j \leq k$ of elements in X. We solve recursively, on the remaining layers, the intersection problem induced by X with target j. The base of the induction is obtained by solving a one-layer Intersection k-Set Cover problem, using the greedy algorithm which provides a $(1+\ln k)$-approximation [26]. We choose the set X and the cardinality j for which we obtain the best ratio of cost to number of covered elements. Next, we include covered elements in the solution under construction, and the problem is reduced consequently.

In order to highlight the main ideas of our approach, we focus on the special case $h = 2$, and we neglect polylogarithmic factors in the analysis.

▶ **Theorem 3.** *There is a $\tilde{O}(\sqrt{k})$-approximation algorithm for* Intersection k-Set Cover *on two layers.*

Proof. Consider the algorithm in Figure 1. Its running time is polynomial, since `SCI` procedure calls the one-layer greedy algorithm $O(Nk^2)$ times.

Let $(\mathcal{O}^1, \mathcal{O}^2) \subseteq \mathcal{S}^1 \times \mathcal{S}^2$ be the optimal solution, and let $K_{\mathcal{O}} \subseteq (\cup_{S\in\mathcal{O}^1} S) \cap (\cup_{S\in\mathcal{O}^2} S)$ be any set of k elements in the intersection. For each element $x \in K_{\mathcal{O}}$ and layer $i = 1, 2$, let us fix a set $\mathcal{O}^i(x) \in \mathcal{O}^i$ that covers x. We prove that at each iteration of the main loop $C'/b' = \tilde{O}(opt/\sqrt{k - |K|})$. This implies that the total cost of the constructed solution is bounded by $\sum_{i=0}^{k-1} \tilde{O}(opt/\sqrt{k-i}) = opt \cdot \tilde{O}(\sqrt{k})$.

Let $\kappa := \sqrt{k - |K|}$. We consider two cases, depending on whether there exists a set X in the optimal solution that covers at least κ elements of $K_{\mathcal{O}} \setminus K$.

Case 1. Assume that there exists $1 \le a \le 2$ and $X \in \mathcal{O}^a$, such that for at least κ elements x of $K_{\mathcal{O}} \setminus K$ we have $\mathcal{O}^a(x) = X$. Let us focus on the moment when our algorithm considers taking the set X. Obviously we have $\kappa \le k - |K|$, therefore our algorithm considers covering $b := \kappa$ elements of X. As the optimal solution does it, it may be done with cost opt, so the call to the one layer algorithm returns a solution with cost $\tilde{O}(opt)$. Hence we have $C'/b' = \tilde{O}(opt/\sqrt{k - |K|})$.

Case 2. For each $1 \le a \le 2$ and every $X \in \mathcal{O}^a$, at most $\kappa - 1$ elements of $K_{\mathcal{O}} \setminus K$ satisfy $\mathcal{O}^a(x) = X$. For each $x \in K_{\mathcal{O}} \setminus K$, let $w(x) := w^1(\mathcal{O}^1(x)) + w^2(\mathcal{O}^2(x))$ be the sum of the costs of sets covering x in the optimal solution. We have

$$\sum_{x\in K_{\mathcal{O}}\setminus K} w(x) = \sum_{a=1}^{2} \sum_{X\in\mathcal{O}^a} \sum_{x\in K_{\mathcal{O}}\setminus K:\ \mathcal{O}^a(x)=X} w^a(\mathcal{O}^a(x)) \le \sum_{a=1}^{2} \sum_{X\in\mathcal{O}^a} w^a(X)\kappa \le \kappa \cdot opt.$$

Thus there exists $x_0 \in K_{\mathcal{O}} \setminus K$ such that $w(x_0) \le \kappa \cdot opt/|K_{\mathcal{O}} \setminus K|$. If we take any a and consider the iteration with $X = \mathcal{O}^a(x_0)$ and $b = 1$, the algorithm computes a set of minimum cost $C_0 \le w(x_0)$ covering x_0. We can conclude that

$$\frac{C'}{b'} \le C_0 \le \frac{\kappa \cdot opt}{|K_{\mathcal{O}} \setminus K|} = \tilde{O}(opt/\sqrt{k - |K|}).$$

◀

It is easy, just more technical, to extend the same approach to $h > 2$ and to refine (slightly) the approximation factor. With some more work, our approach generalizes to Intersection k-Nonmetric Facility Location. The details of this generalization will be given in the full version of this paper.

▶ **Theorem 4.** *There exists a $(4k^{1-1/h}\log^{1/h}(k))$-approximation algorithm for* Intersection k-Nonmetric Facility Location *(hence for* Intersection k-Set Cover*) running in $N^{O(h)}$ time.*

2.3 Intersection k-MST

Our Intersection k-MST algorithm works as follows. We consider a new metric w defined as $w(e) := \sum_i w^i(e)$ for each $e \in E$, and compute a 2-approximate solution of the resulting (one-layer) k-MST problem using the algorithm in [12].

▶ **Lemma 5.** *Let $K \subseteq V$, and $w^i(K)$ denote the cost of a minimum spanning tree of K on layer i. Then there exist two nodes $u, v \in K$ such that $w^i(u, v) \le 4w^i(K)/|K|^{1/h}$ for $i = 1, \ldots, h$.*

Proof. Let us prove the following claim by induction on i: for any $i \in \{0, \dots, h-1\}$, there exist a nodeset $K_i \subseteq K$ and paths $P_i^1, P_i^2 \dots, P_i^i$ on K_i such that: (a) $|K_i| \geq |K|^{1-i/h}$ and (b) $w^j(P_i^j) \leq 2w^j(K)/|K|^{1/h}$ for $j = 1, \dots, i$. Trivially $K_0 = K$ satisfies the claim, hence assume $i > 0$. Let T^i be the minimum spanning tree of K on layer i. Duplicate its edges, compute an Euler tour, and shortcut duplicated nodes. Let C^i be the resulting cycle on K of length at most $2w^i(K)$. Remove up to $|K|^{1/h}$ edges from C^i so as to obtain $|K|^{1/h}$ segments of length at most $2w^i(K)/|K|^{1/h}$ each. Let P be the segment maximizing the cardinality of $K_i := V(P) \cap K_{i-1}$. Set K_i satisfies (a) since $|K_i| \geq |K_{i-1}|/|K|^{1/h} \geq |K|^{1-(i-1)/h-1/h}$. The paths P_i^i and P_i^j, $j < i$, satisfying (b) are obtained from P and P_{i-1}^j, respectively, by shortcutting the nodes not in K_i.

Similarly as above, we can split C^h into $|K|^{1/h}/2$ segments which span K and have length at most $4w^h(K)/|K|^{1/h}$ each. At least one of these segments contains $2|K_{h-1}|/|K|^{1/h} \geq 2$ nodes of K_{h-1}. Thus there are two nodes u and v such that $w^i(u,v) \leq 4w^i(K)/|K|^{1/h}$ for $i = 1, \dots, h$. ◀

▶ **Theorem 6.** *The* Intersection k-MST *algorithm above is* $16k^{1-1/h}$*-approximate.*

Proof. Consider the following process: starting with the optimal set $K_{\mathcal{O}}$ of k covered nodes, we iteratively take the edge $\{x, y\}$ guaranteed by Lemma 5 and contract it in all layers, until $K_{\mathcal{O}}$ collapses into a single node. The contracted edges form a tree T' (same for all layers) spanning k nodes, of cost

$$w(T') \leq 4 \sum_{i=1}^{h} w^i(K_{\mathcal{O}}) \sum_{i=1}^{k-1} (k-i+1)^{-\frac{1}{h}} \leq \; 8k^{1-\frac{1}{h}} \sum_{i=1}^{h} w^i(K_{\mathcal{O}}) = 8k^{1-\frac{1}{h}} opt.$$

The algorithm returns a solution of cost at most $2w(T')$. The claim follows. ◀

Via a non-trivial construction we can show that the approximation factor of our algorithm is $\Omega(\frac{1}{h}k^{1-1/h})$. The details will be given in the full version of this paper.

3 Union Problems

In this section we present our results for Union k-MST and Union k-Metric Facility Location. In (unrooted) Union k-MST we have the same input and output as in Intersection k-MST, but here the trees T^i must satisfy $|\bigcup_i V(T^i)| \geq k$. In the rooted version of the problem, we are also given a root r^i for each layer i with the constraint $r^i \in T^i$. The Union MST problem is the special case of Union k-MST with $k = n$ (rooted and unrooted versions are equivalent). In Union k-Metric Facility Location we have the same input and output as in Intersection k-Metric Facility Location, but the objective function to be minimized is $\sum_i \sum_{f \in \mathcal{A}^i} o^i(f) + \sum_{c \in \mathcal{C}'} \min_i \{w^i(c, \mathcal{A}^i)\}$.

3.1 Rooted Union k-MST

▶ **Theorem 7.** *Rooted* Union k-MST *and* Union k-Metric Facility Location *are APX-hard for any* $h \geq 1$. Union MST *is APX-hard for any* $h \geq 2$.

Proof. The first claim trivially follows from the APX-hardness [12, 17] of the considered problems for $h = 1$, by adding dummy layers with infinite edge weights.

For the second claim, we consider a reduction from the APX-hard [5] Prize-Collecting Steiner Tree problem: given an undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_{\geq 0}$, a root node $r \in V$, and node prizes $p : V \to \mathbb{R}_{\geq 0}$, find a tree $T \ni r$ which minimizes

$\sum_{e\in T} w(e) + \sum_{v\notin T} p(v)$. We create a first layer, with edge weights $w^1 = w$. Then we construct a second layer, where we set $w^2(\{r, v\}) = p(v)$ for any $v \in V$. All the other layers, if any, are dummy layers defined as above. This reduction is approximation preserving. ◀

▶ **Theorem 8.** *For an arbitrary number of layers, rooted* Union k-MST *and* Union k-Metric Facility Location *are not approximable better than* $\Omega(\log k)$ *unless* $P = NP$*, even when* $k = n$*.*

Proof. We prove the claim for rooted Union k-MST, by giving a reduction from cardinality Set Cover: given a universe $\mathcal{U}$ of n' elements, and a collection $\mathcal{S} = \{S_1, \ldots, S_{m'}\}$ of m' subsets of $\mathcal{U}$, find a minimum cardinality subset $\mathcal{A} \subseteq \mathcal{S}$ which spans $\mathcal{U}$. This problem is $\Omega(\log n')$-hard to approximate [24]. We create one node per element of $\mathcal{U}$, plus two extra nodes r and s. We create one layer i for each set S_i (i.e., $h = m'$). In layer i we let $w^i(\{r, s\}) = 1$ and $w^i(\{s, v\}) = 0$ for each $v \in S_i$. We also let $r^i := r$ for each i, and assume $k = n = n' + 2$. Note that any solution to the rooted Union k-MST instance of cost α can be turned into a solution to the Set Cover instance of the same cost, and vice versa.

To prove the claim for Union k-Metric Facility Location, we use the same reduction as above, where the edge $\{r, s\}$ is replaced by a single node r, which is a facility of opening cost 1. ◀

A simple greedy algorithm guarantees a $O(\log k)$-approximation which matches the above lower bound. The basic idea is as follows. Suppose that the considered covering problem satisfies a natural *composition* property, namely two solutions satisfying k' and k'' distinct requests, can be merged (without increasing the total cost) to obtain a solution satisfying $k' + k''$ requests. (Merging might involve some polynomial-time operations). Suppose also that there exists a ρ-approximation for one layer. The idea is then to compute, for each layer separately and for each $k' \leq k$, a ρ-approximate solution to the partial covering instance induced by that layer with target k'. The solution providing the best ratio of cost to number k' of satisfied requests is *merged* with the solution under construction. Then satisfied requests are *removed* from the set of requests, k decremented by k', and the process is iterated until $k \leq 0$. Via standard techniques this algorithms provides a $O(\rho \cdot \log k)$-approximation.

Removing requests transforms a given k-MST instance in each layer into a k-Steiner Tree instance: for the latter problem there is a 4-approximation algorithm [12]. Note also that all the partial solutions in each layer contain the corresponding root r^i: hence the merging step is trivial. For k-Metric Facility Location, there is a 2-approximation algorithm in [19]. In this case removing a request simply means removing one client, and the merging step is trivial. Altogether:

▶ **Theorem 9.** *There exist* $O(\log k)$*-approximation algorithms for* Union k-MST *and* Union k-Metric Facility Location*.*

We next describe an LP-based $O(h)$-approximation algorithm for rooted Union k-MST. This is an improvement over the $\Theta(\log k)$-approximation given by the greedy algorithm for the relevant case of bounded h.

For notational convenience, we assume that the roots $R := \cup_i \{r^i\}$ are not counted into the target number k of connected nodes. In other terms, we replace k by $k - |R|$. We make the same assumption also in the case of one layer. Consider the following LP relaxation for

k-Steiner Tree ($W \ni r$ is the set of terminals) denoted by $LP_{kST}(w, W, V, r, k)$:

$$\begin{array}{lll} \min & \sum_{e\in E} w(e)\, x_e & \\ s.t. & \sum_{e\in\delta(S)} x_e \geq z_v, & \forall (v,S) : S \subseteq V - \{r\}, v \in S \cap W; \\ & \sum_{v\in W} z_v \geq k; & \\ & x_e \geq 0, 1 \geq z_v \geq 0, & \forall v \in W, \forall e \in E. \end{array}$$

Here, variable x_e indicates whether edge e is included in the solution, whereas variable z_v indicates whether terminal v is connected. Moreover $\delta(S)$ denotes the set of edges with exactly one endpoint in S. Observe that $LP_{kMST}(w, V, r, k) := LP_{kST}(w, V, V, r, k)$ is an LP relaxation for k-MST. We need the following lemmas.

▶ **Lemma 10.** *[11] Let (w, V, r, k) be an instance of k-MST, $w_{max} := \max_{v\in V}\{w(r, v)\}$, and opt′ be the optimal solution to $LP_{kMST}(w, V, r, k)$. There is a polynomial-time algorithm* `apx-kmst` *which computes a solution to the instance of cost at most $2opt' + w_{max}$.*

▶ **Lemma 11.** *[10] Let $G = (V\cup\{v\}, E)$ be a directed graph, with edge capacities $\alpha : E \to \mathbb{R}_{\geq 0}$ such that $\sum_{e\in\delta^+(u)} \alpha(e) = \sum_{e\in\delta^-(u)} \alpha(e)$ for all $u \in V \cup \{v\}$. Then there is a pair of edges (u, v) and (v, z), such that the following capacity reservation β supports the same flow as α between any pair of nodes in V: for $\Delta\alpha := \min\{\alpha(u, v), \alpha(v, z)\}$, set $\beta(u, v) = \alpha(u, v) - \Delta\alpha$, $\beta(v, z) = \alpha(v, z) - \Delta\alpha$, $\beta(u, z) = \alpha(u, z) + \Delta\alpha$, and $\beta(e) = \alpha(e)$ for the remaining edges e.*

▶ **Corollary 12.** *Given a feasible solution (x, z) to $LP_{kST}(w, W, V, r, k)$, there is a feasible solution (x', z') to $LP_{kMST}(w, W, r, k)$ such that $\sum_e w(e)x'_e \leq 2 \cdot \sum_e w(e)x_e$.*

Proof. Variables x_e can be interpreted as a capacity reservation which supports a fractional flow of value z_v from each $v \in W$ to the root. Let us replace each edge with two oppositely directed edges, and assign to each such edge the same weight and capacity as the original edge. This way, we obtain a capacity reservation α which costs twice the original capacity reservation, and satisfies the condition of Lemma 11. We consider any non-terminal node $v \neq r$ with some incident edge of positive capacity, and apply Lemma 11 to it. Due to triangle inequality, the cost of the capacity reservation does not increase. We iterate the process on the resulting capacity reservation. Within a polynomial number of steps, we obtain a capacity reservation β which: (1) supports the same flow from each terminal to the root r as α, (2) has value 0 on edges incident to non-terminal nodes (besides r), and (3) does not cost more than α. At this point, we remove the nodes $V - (W \cup \{r\})$, and merge the capacity of oppositely directed edges to get an undirected capacity reservation x'. By construction, the pair (x', z) is a feasible solution to $LP_{kMST}(w, W, r, k)$ of cost at most $2 \cdot \sum_e w(e)x_e$. ◀

We are now ready to describe our algorithm for rooted Union k-MST. In a preliminary step we guess the largest distance L in the optimal solution between any connected node and the corresponding root, and discard nodes at distance larger than L from their root. This introduces a factor $O(nh)$ in the running time. Note that $L \leq opt$. We let V^i be the remaining nodes in layer i.

Then we compute the optimal fractional solution $OPT^* = (x^i, z^i, z)_i$, of cost opt^*, to the following LP relaxation LP_{ukMST} for the problem, where variables x^i_e and z^i_v indicate whether edge e is included in the solution of layer i and node v is connected in layer i, respectively. Variable z_v indicates whether node v is connected in at least one layer.

$$
\begin{array}{lll}
\min & \sum_{i=1,\ldots,h} \sum_{e\in E} w^i(e)\, x^i_e & \\
s.t. & \sum_{e\in\delta(S)} x^i_e \geq z^i_v, & \forall i \in \{1,\ldots,h\}, \forall (v,S) : S \subseteq V^i - \{r^i\}, v \in S; \\
& \sum_{i=1,\ldots,h} z^i_v \geq z_v, & \forall v \in V - R; \\
& \sum_{v\in V-R} z_v \geq k; & \\
& z^i_v, x^i_e \geq 0, 1 \geq z_v \geq 0, & \forall i \in \{1,\ldots,h\}, \forall v \in V - R, \forall e \in E.
\end{array}
$$

Given OPT^*, we compute for each layer i a subset of nodes W^i, where v belongs to W^i iff $z^i_v = \max_{j=1,\ldots,h}\{z^j_v\}$ (breaking ties arbitrarily). We also define $k^i := \lfloor \sum_{v\in W^i} z_v \rfloor$. For each layer i, we consider the k-MST instance on nodes $W^i \cup \{r^i\}$ with target k^i. This instance is solved using the 2-approximation algorithm `apx-kmst` of Lemma 10: the resulting tree T^i is added to the solution for layer i. Let k' be the number of connected nodes. If $k' < k$, the algorithm connects $k - k'$ extra nodes, chosen greedily, to the corresponding root in order to reach the global target k.

▶ **Theorem 13.** *There is a $O(h)$-approximation algorithm for rooted* Union k-MST. *The running time of the algorithm is $O((nh)^{O(1)})$.*

Proof. Consider the above algorithm. The claim on the running time is trivial. By construction, the solution computed is feasible (i.e., it connects k nodes). It remains to consider the approximation factor.

For each $v \in W^i$, we let $\tilde{z}^i_v = z_v$, and set $\tilde{z}^i_v = 0$ for the remaining nodes. Furthermore, we let $\tilde{x}^i_e = h \cdot x^i_e$. Observe that $(\tilde{x}^i, \tilde{z}^i, z)_i$ is a feasible fractional solution to LP_{ukMST} of cost $h \cdot opt^*$. Observe also that $(\tilde{x}^i, \tilde{z}^i)$ is a feasible solution to $LP_{kST}(w^i, W^i, V^i, r^i, k^i)$: let $a\tilde{p}x^i$ be the associated cost. By Lemma 11, there is a fractional solution to $LP_{kMST}(w^i, W^i, r^i, k^i)$ of cost at most $2a\tilde{p}x^i$. It follows from Lemma 10 that the solution computed by `apx-kmst` on layer i costs at most $4a\tilde{p}x^i + L$.

Since the W^i's are disjoint, the algorithm initially connects at least $\sum_i k^i \geq k - h$ nodes. Hence the cost of the final augmentation phase is at most $h \cdot L \leq h \cdot opt$. Putting everything together, the cost of the solution returned by the algorithm is at most:

$$\sum_i (4 \cdot a\tilde{p}x^i + L) + h \cdot L \leq 4h \cdot opt^* + 2h \cdot L \leq 6h \cdot opt\,.$$

The constant multiplying h in the approximation factor can be reduced with a more technical analysis, at the cost of a higher running time. We also observe that the integrality gap of LP_{ukMST} is $\Omega(h)$. In fact, consider the (cardinality) Set Cover instance as in [27]: given a hyper-graph on m' nodes, with hyper-edges given by all subsets of $m'/2$ nodes, create an element for each hyper-edge, and a set for each node containing all the hyper-edges incident to that node. It is easy to see that the best fractional solution for the standard set cover LP (assigning value $2/m'$ to all elements) has cost 2, while the best integral solution contains $m'/2 + 1$ sets. The same reduction as in Theorem 8 implies the claim.

Essentially the same approach works also for Union k-Metric Facility Location. Also in this case, we can show that the corresponding LP has integrality gap $\Omega(h)$.

▶ **Theorem 14.** *There is a $O(h)$-approximation algorithm for* Union k-Metric Facility Location. *The running time of the algorithm is $O((nh)^{O(1)})$.*

3.2 Unrooted Union k-MST

▶ **Theorem 15.** *Unrooted* Union k-MST *is not approximable in polynomial time for an arbitrary number h of layers unless $P = NP$.*

Proof. We give a reduction from SAT: given a CNF boolean formula on m' clauses and n' variables, determine whether it is satisfiable or not. For each variable i, we create two nodes t_i and f_i. Intuitively, these nodes represent the fact that i is true or false, respectively. Furthermore, we have a node for each clause. Hence the overall number of nodes is $n = 2n' + m'$. We create a separate layer for each variable i (i.e., $h = n'$). In layer i, we connect with an edge of cost zero t_i (resp., f_i) to all the clauses which are satisfied by setting i to true (resp., to false)[3]. The target value is $k = n' + m'$. Note that, there is a satisfying assignment to the SAT instance iff there is a solution of cost zero to the Union k-MST instance. ◀

For $h = O(1)$, the rooted and the unrooted versions of the problem are equivalent approximation-wise. In fact, one obtains an approximation-preserving reduction from the unrooted to the rooted case by guessing one node r^i in the optimal solution per layer: this introduces a polynomial factor $O(n^h)$ in the running time. We remark that an exponential dependence on h of the running time is unavoidable in the unrooted case, due to Theorem 15. An opposite reduction is obtained by appending n dummy nodes to each root (distinct nodes for distinct layers), with edges of cost zero, and setting the target to $k + hn$. The following result follows.

▶ **Corollary 16.** *Unrooted* Union k-MST *is APX-hard for any $h \geq 1$. There is a $O(h)$-approximation algorithm for the problem of running time $O((hn)^{O(1)}n^h)$.*

4 Conclusions and Open Problems

In this paper, we introduced multi-layer covering problems, a new framework that can be used to describe a wide spectrum of yet unstudied problems. We addressed two natural ways of combining the layers: intersection and union. We gave multi-layer approximation algorithms, as well as hardness results, for a few classic covering problems (and their partial covering versions). There are several research questions that merit further study.

- There are other natural ways one can combine the layers. Consider, for example, the car/bike problem in the case where you can put your bike in the car trunk. Now you can make more than one tour by bike, the only requirement being that the bike tours all touch the (unique) car tour.
- What about min-max multi-layer problems, where the goal is to minimize the maximum cost over the layers?
- We considered covering problems: what about packing problems?
- Our algorithms for union problems give tight bounds only with respect to the corresponding natural LPs. This leaves room for improvement.
- There is a considerable gap between upper and lower bounds for intersection problems. In particular, our hardness results do not depend on h, while the approximation ratios deteriorate rather rapidly for increasing h.

[3] Without loss of generality, we can assume that each clause does not contain both a literal and its negation.

References

1 A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Power in unity: forming teams in large-scale community systems. In *CIKM*, pages 599–608, 2010.

2 B. Applebaum, B. Barak, and A. Wigderson. Public-key cryptography from different assumptions. In *STOC*, pages 171–180, 2010.

3 S. Arora, B. Barak, M. Brunnermeier, and R. Ge. Computational complexity and information asymmetry in financial products (extended abstract). In *ICS*, pages 49–65, 2010.

4 A. Berger, V. Bonifaci, F. Grandoni, and G. Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128:355–372, 2011.

5 M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.

6 A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities – an $O(n^{1/4})$ approximation for densest k–subgraph. In *STOC*, pages 201–210, 2010.

7 J. Edmonds. *Matroid intersection*. North-Holland, 1979.

8 F. Eisenbrand, F. Grandoni, T. Rothvoß, and G. Schäfer. Connected facility location via random facility sampling and core detouring. *Journal of Computer and System Sciences*, 76:709–726, 2010.

9 L. Fleischer, J. Könemann, S. Leonardi, and G. Schäfer. Simple cost sharing schemes for multicommodity rent-or-buy and stochastic Steiner tree. In *STOC*, pages 663–670, 2006.

10 A. Frank. On connectivity properties of Eulerian digraphs. *Annals of Discrete Mathematics*, 41:179–194, 1989.

11 N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *FOCS*, pages 302–309, 1996.

12 N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *STOC*, pages 396–402, 2005.

13 F. Grandoni and G. F. Italiano. Improved approximation for single-sink buy-at-bulk. In *ISAAC*, pages 111–120, 2006.

14 F. Grandoni, R. Ravi, and M. Singh. Iterative rounding for multi-objective optimization problems. In *ESA*, pages 95–106, 2009.

15 F. Grandoni and R. Zenklusen. Approximation schemes for multi-budgeted independence systems. In *ESA (1)*, pages 536–548, 2010.

16 Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanita. From uncertainty to nonlinearity: Solving virtual private network via single-sink buy-at-bulk. *Mathematics of Operations Research*, 36(2):185–204, 2011.

17 S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *SODA*, pages 649–657, 1998.

18 Mohammad Taghi Hajiaghayi and Kamal Jain. The prize-collecting generalized steiner tree problem via a new approach of primal-dual schema. In *SODA*, pages 631–640, 2006.

19 K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

20 Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. The matroid median problem. In *SODA*, 2011. To appear.

21 T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476, 2009.

22 C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of Web sources. In *FOCS*, pages 86–92, 2000.

23 R. Ravi and M. X. Goemans. The constrained minimum spanning tree problem (extended abstract). In *SWAT*, pages 66–75, 1996.

24 R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, pages 475–484, 1997.

25 M. Riaz, R. Nielsen, J. Pedersen, N. Prasad, and O. Madsen. A framework for planning a unified wired and wireless ict infrastructure. *Wireless Personal Communications*, 54:169–185, 2010.

26 P. Slavik. Improved performance of the greedy algorithm for partial cover. *Information Processing Letters*, 64(5):251–254, 1997.

27 V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

Tight Gaps for Vertex Cover in the Sherali-Adams SDP Hierarchy*

Siavosh Benabbas[1], Siu On Chan[2], Konstantinos Georgiou[3], and Avner Magen[1]

1 **Department of Computer Science, University of Toronto**
siavosh@cs.toronto.edu
2 **Department of Computer Science, University of California, Berkeley**
siuon@cs.berkeley.edu
3 **Department of Combinatorics and Optimization, University of Waterloo**
k2georgiou@math.uwaterloo.ca

Abstract

We give the first tight integrality gap for Vertex Cover in the Sherali-Adams SDP system. More precisely, we show that for every $\epsilon > 0$, the standard SDP for Vertex Cover that is strengthened with the level-6 Sherali-Adams system has integrality gap $2-\epsilon$. To the best of our knowledge this is the first nontrivial tight integrality gap for the Sherali-Adams SDP hierarchy for a combinatorial problem with hard constraints.

For our proof we introduce a new tool to establish Local-Global Discrepancy which uses simple facts from high-dimensional geometry. This allows us to give Sherali-Adams solutions with objective value $n(1/2 + o(1))$ for graphs with small $(2 + o(1))$ vector chromatic number. Since such graphs with no linear size independent sets exist, this immediately gives a tight integrality gap for the Sherali-Adams system for superconstant number of tightenings. In order to obtain a Sherali-Adams solution that also satisfies semidefinite conditions, we reduce semidefiniteness to a condition on the Taylor expansion of a reasonably simple function that we are able to establish up to constant-level SDP tightenings. We conjecture that this condition holds even for superconstant levels which would imply that in fact our solution is valid for superconstant level Sherali-Adams SDPs.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Vertex Cover, Integrality Gap, Lift-and-Project systems, Linear Programming, Semidefinite Programming

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.41

1 Introduction

A vertex cover of a graph $G = (V, E)$ is a subset S of the vertices such that for every edge $ij \in E$ at least one vertex among i, j lies in S. In the Minimum Vertex Cover problem the objective is to find the vertex cover of minimum size. While a 2-approximation algorithm is rather straightforward, considerable effort has failed to yield any polynomial time algorithm with approximation ratio $2-\Omega(1)$. Indeed the best algorithm known achieves an approximation ratio of $2-O(\sqrt{1/\log n})$ [21]. On the other hand, the strongest PCP-based hardness result [12] shows that 1.36-approximating Vertex Cover is **NP**-hard. Only by

* The full version of this work is available as [5]

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 41–54

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

assuming Khot's Unique Game Conjecture [24], whose validity is the subject of an active area of research (see [1, 25] for example), one can show a $2 - o(1)$ hardness.

Motivation for studying Vertex Cover is two-fold. For one thing it is arguably one of the simplest **NP**-hard problems whose inapproximability remains unresolved. But more importantly, studying Vertex Cover has introduced some very important techniques both in terms of approximation algorithms and hardness of approximation with [12] being a prime example. Intuitively this is, at least partly, due to the "hard constraints" of Vertex Cover, that is the solution has to satisfy a number of inflexible constraints (the edge constraints). As many of the standard techniques for proving hardness of approximation and integrality gaps produce solutions which satisfy *most* constraints in an instance, showing tight hardness for Vertex Cover has remained unresolved.

Trying to resolve the approximability of Vertex Cover, one could study the behavior of prominent algorithmic schemes, such as Linear Programming (LP) and Semidefinite Programming (SDP) relaxations, which have yielded state-of-the-art algorithms for many combinatorial optimization problems. There, the measure of efficiency is the *Integrality Gap* which sets the approximation limitation of the algorithms based on these relaxations. In this work we show that a large family of LP and SDP relaxations for Vertex Cover have integrality gap arbitrarily close to 2. Such an integrality gap rules out a rich and important family of approximation algorithms for the problem at hand.

Furthermore, there seems to be a connection between integrality gaps for strong LP/SDP relaxations of a problem and its hardness of approximation. In one direction the reductions used to establish hardness of approximation for many problems have been used to construct integrality gaps for them, e.g. [26, 9, 29, 33]. In the other direction, and specifically for Vertex Cover, Vishwanathan [34] shows that any hard instance of the problem should have subgraphs that look like the so called "Borsuk graphs". Interestingly a specific subfamily of Borsuk graphs were previously used in many integrality gap instances for Vertex Cover, e.g. [17, 8, 20, 15, 16]. To make the picture even more complete, we show that *any* Borsuk graph is a good integrality gap instance for the (so called) Sherali-Adams LP system of relaxations for Vertex Cover.

The (tight) integrality gap of the standard LP and SDP relaxations for Vertex Cover has long been resolved [17]. Nevertheless, celebrated relaxations for a number of combinatorial problems require strengthenings (addition of extra constraints) aiming to drop the integrality gap. In that direction, a number of systematic procedures, known as Lift-and-Project systems have been proposed to systematically improve the integrality gap. These systems build strong hierarchies of either LP relaxations (as the Lovász-Schrijver and the Sherali-Adams systems) or SDP relaxations (as the Lovász-Schrijver SDP, the Sherali-Adams SDP and the Lasserre systems). Lift-and-Project systems can be thought of as being applied in rounds (also called levels). The bigger the number of rounds used, the more accurate the obtained relaxation is. In fact, if as many rounds as the number of variables are used, the final relaxation is exact and no integrality gap exists. On the other hand the size of the derived relaxation grows exponentially with the number of rounds, which implies that the time one needs to solve it also grows. It is then natural to ask whether looking at a modest number of rounds (say $O(1)$ or $\log \log n$) will result in an algorithm with approximation factor better than 2.

Identifying the limitations of relaxations derived by Lift-and-Project system has attracted much attention and showing integrality gaps for the Sherali-Adams SDP and the Lasserre systems stand as the most attractive subjects in this area of research due to a number of reasons. Firstly, the best algorithms known for many combinatorial optimization problems

(and Vertex Cover in particular) are based on relaxations weaker than those derived by a constant (say four) rounds of the Sherali-Adams SDP system which we study here, e.g. [18, 23, 2, 21]. Lift-and-Project hierarchies have been also used recently in designing approximation algorithms with a runtime-approximation ratio trade off, e.g. [11, 27, 10, 4, 22, 3, 19]. Finally, for some particular constraint satisfaction problems, and modulo the Unique Games Conjecture, no approximation algorithm can perform better than the one obtained by Sherali-Adams SDP of a constant number of rounds (see [28].) One can then think of algorithms based on the Sherali-Adams SDP as an interesting model of computation.

In this work we study the limitations of strong relaxations for Vertex Cover in the powerful Sherali-Adams SDP system. The performance of the same hierarchy has been studied for other combinatorial problems (see [29, 6, 7]), but its integrality gap for Vertex Cover remained open, due to the hard constraints mentioned earlier. Our main result is as follows.

▶ **Theorem 1.1.** *For every $\epsilon > 0$, the SDP derived by the level-6 Sherali-Adams SDP system for* Vertex Cover *has integrality gap $2 - \epsilon$.*

Theorem 1.1 yields the first nontrivial Sherali-Adams SDP integrality gap for Vertex Cover and in fact any problem with hard constraints. While tight integrality gaps for weaker or incomparable systems were known, there were no good candidates for Sherali-Adams SDP integrality gap solutions.In particular, while integrality gaps for the closely related but weaker Sherali-Adams LP system for Vertex Cover were known [9], the solution there does *not* satisfy the required positive semidefiniteness condition. As we explain below, apart from the significance of our new SDP integrality gap, we also believe that our proofs are interesting in their own right. In Section 3 we give a high level description of our ideas, along with a detailed explanation of how our techniques are different from existing integrality gap results.

On our way to prove the above theorem we need to define new solutions for Sherali-Adams LP relaxations of Vertex Cover. As mentioned, one of our contributions is an intuitive and geometric explanation of why this large family of LPs are fooled by a certain family of graphs, the so-called Borsuk graphs. This yields a tight level-$\Omega(\sqrt{\log n / \log\log n})$ integrality gap for Sherali-Adams LP (see Theorem 4.4.) Other than being used in our proof of Theorem 1.1, our solution is arguably simpler and more intuitive than the integrality gap of [9] for the same system.[1]

The heart of the problem in showing integrality gaps for Sherali-Adams SDPs is that the proposed solution needs to satisfy a strong positive-semidefiniteness condition. Toward establishing Theorem 1.1, we show how to reduce this condition into a clean analytic statement about a certain function parameterized by t. We are able to show that this analytic statement holds up to $t = 6$, hence the level-6 Sherali-Adams SDP gap. We have strong evidence (both theoretical and experimental) that the aforementioned analytic statement holds for any constant value of t, which we explicitly state as a conjecture in Section 5.3. To sum up, we have the following second theorem.

▶ **Theorem 1.2.** *Assuming Conjecture 5.12, for every constant $\epsilon > 0$ and $t \in \mathbb{N}$, the SDP derived by the level-t Sherali-Adams SDP system for* Vertex Cover *has integrality gap $2 - \epsilon$.*

For a brief discussion of the validity of Conjecture 5.12 see Remark 5.3.

[1] Although it should be mentioned that their integrality gap applies to more rounds.

Known integrality gaps for Vertex Cover: Considerable effort has been invested in strong lower bounds for various hierarchies for Vertex Cover. For LP hierarchies, [31] shows an integrality gap of $2-\epsilon$ for $\Omega(n)$ rounds of the Lovász-Schrijver system and [9] shows the same integrality gap for the stronger Sherali-Adams system up to $\Omega(n^\delta)$ rounds (with δ going to 0 together with ϵ.) Both results concern LP hierarchies, which are incomparable to SDP relaxations. For SDP hierarchies, and for the Lovász-Schrijver SDP system which is stronger than both the LS system and the canonical SDP formulation (but incomparable to Sherali-Adams), [14] shows an integrality gap of $2-\epsilon$ for $\Omega(\sqrt{\log n/\log\log n})$ levels.

The integrality gap of two stronger hierarchies for Vertex Cover, on the other hand, has long been open. The first is the Sherali-Adams SDP system which is stronger than the LS system, and the subject of this paper. The second is the Lasserre system, for which no tight integrality gap for Vertex Cover is known.[2] If one is content with an integrality gap less than 2, a 1.36 integrality gap for $\Omega(n^\delta)$ levels [33] and a 7/6 integrality gap for $\Omega(n)$ levels [30] of the Lasserre system are known. We will compare our proof techniques with previous ones at the end of Section 3.

2 Preliminaries

2.1 Borsuk Graphs, Frankl-Rödl Graphs and Tensoring

Our integrality gap instances are *Frankl-Rödl graphs*. These graphs are parameterized by an integer m which is considered growing and a real parameter $0 < \gamma < 1$.

▶ **Definition 2.1.** (*Frankl-Rödl graphs*) The *Frankl-Rödl graph* G^m_γ is the graph with vertices $\{-1,1\}^m$ where two vertices $i, j \in \{-1,1\}^m$ are adjacent iff $d_H(i,j) = (1-\gamma)m$.

Frankl-Rödl graphs exhibit an interesting "extremal" combinatorial property. While G^m_0 is a perfect matching and thus has a vertex cover of size half the number of its vertices, a beautiful theorem by Frankl and Rödl states that for slightly larger γ, any vertex cover of G^m_γ is very large. The fact that such a small geometric perturbation results in a drastic change in the vertex cover size has led to the use of Frankl-Rödl graphs as tight integrality gap instances in a series of results [17, 8, 14, 15, 16].

▶ **Theorem 2.2** ([14]; slight modification of Theorem 1.4 of [13])**.** *Let m be an integer and let $\gamma = \Theta(\sqrt{\log m/m})$ be a sufficiently small number so that γm is an even integer. Then any vertex cover of G^m_γ contains at least a $1-o(1)$ fraction of the vertices.*

An important tool in proving strong integrality gaps is tensoring of vectors. Recall that for $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^m$ their *tensor product* $\mathbf{u} \otimes \mathbf{v} \in \mathbb{R}^{nm}$ is a vector indexed by ordered pairs from $[n] \times [m]$ taking value $u_i v_j$ at coordinate (i,j). For any polynomial $P(x) = c_1 x^{t_1} + \ldots + c_q x^{t_q}$ *with nonnegative coefficients* consider the function T_P mapping a vector $\mathbf{u} \in \mathbb{R}^n$ to the vector $T_P(\mathbf{u}) = (\sqrt{c_1}\mathbf{u}^{\otimes t_1}, \ldots, \sqrt{c_q}\mathbf{u}^{\otimes t_q}) \in \mathbb{R}^{\sum_i n^{t_i}}$, where $\mathbf{u}^{\otimes d}$ is the vector obtained by tensoring $\mathbf{u}$ with itself d times. Polynomial tensoring can be used to manipulate inner products in the sense that $T_P(\mathbf{u}) \cdot T_P(\mathbf{v}) = P(\mathbf{u} \cdot \mathbf{v})$; it was used as an ingredient in many integrality gap results such as [17, 8, 14, 15].

We often think of the vertices of the Frankl-Rödl graphs as (scaled and) embedded on the unit sphere S^{m-1}. In this sense the Frankl-Rödl graphs are subgraphs of the infinite *Borsuk graphs*.

[2] In fact there are only a few combinatorial problems for which tight Lasserre integrality gaps are known. (see [30] and [33] for some notable exceptions.)

▶ **Definition 2.3.** (*Borsuk graphs*) The Borsuk graph B_δ^m is an infinite graph with vertex set S^{m-1}. Two vertices $\mathbf{x}, \mathbf{y}$ are adjacent if they are nearly antipodal, i.e. $\|\mathbf{x}+\mathbf{y}\| \leq 2\sqrt{\delta}$.

2.2 Strong relaxations for Vertex Cover

In this subsection we give a brief high level description of the Sherali-Adams SDP system applied to the Vertex Cover problem. This high level description should be enough to understand the high level of our results. The interested reader can find a rigorous definition in the full version of the paper or [32].

The starting point of the Sherali-Adams SDP for Vertex Cover is the following simple LP relaxation of Vertex Cover. Assume that $G = (V, E)$ is the input graph.

$$\min \sum_{i \in V} x_i, \qquad \text{s.t. } \forall ij \in E \;\; x_i + x_j \geq 1, \qquad \forall i \in V \;\; x_i \in [0,1] \tag{1}$$

Here x_i is the indicator variable of vertex i being part of a vertex cover. Since in the LP relaxation (1) x_i assumes any value in $[0,1]$, we may think of x_i as encoding a local distribution $\mathcal{D}(\{i\})$ of 0-1 assignments for the elements in $\{i\}$. The Sherali-Adams LP system strengthens this relaxation by introducing variables to encode the joint status of a subset of vertices U with respect to the vertex cover, for all subsets up to a certain size. In particular, the Sherali-Adams LP system of level t, seen below, is a Linear Program with the following variables. If $U \subset V$ is any subset of the vertices of size at most t, the program will have real-valued variables to specify a distribution $\mathcal{D}(U)$ over the subsets of U. Furthermore, the program will have two kinds of constraints. The first kind (similar to the one in (1)) ensure that any subset of U that is assigned a positive probability covers all the edges inside U, i.e. the distribution $\mathcal{D}(U)$ is over vertex covers of U. The second kind of constraints ensure that the marginals of the distributions for $U_1 \subseteq U$ are consistent on U_1, i.e. any event that only depends on the vertices of U_1 has the same probability according to $\mathcal{D}(U_1)$ and $\mathcal{D}(U)$. The objective value of the program is the sum over all vertices v, of the probability that v is in the local vertex covers (which is well defined as $\mathcal{D}(U)$'s are consistent for all $U \ni v$.) That is, fix a $U \ni v$, the contribution of v to the objective function is, $\mathbb{P}_{S \sim \mathcal{D}(U)}[v \in S]$. Summarizing we have the following relaxations,

▶ **Definition 2.4** (Level-t Sherali-Adams LP relaxation of Vertex Cover)**.** Let $\mathcal{P}(U)$ denote the powerset of U.

$$\begin{array}{lll} \min & \sum_{i \in V} \mathbb{P}_{S \sim \mathcal{D}(\{i\})}[i \in S] & \\ \text{s.t.} & \mathbb{P}_{S \sim \mathcal{D}(\{i,j\})}[i \notin S, j \notin S] = 0 & \forall ij \in E \qquad \textit{(Edge constraints)} \\ & \mathbb{P}_{S \sim \mathcal{D}(U_1)}[S = T] = \mathbb{P}_{S \sim \mathcal{D}(U)}[S \cap U_1 = T] & \forall T \subseteq U_1 \subseteq U \subseteq V, |U| \leq t \\ & \mathcal{D}(U) \text{ is a distribution on } \mathcal{P}(U) & \forall U \subseteq V, |U| \leq t \end{array} \tag{2}$$

▶ **Definition 2.5** (Level-t Sherali-Adams SDP relaxation of Vertex Cover)**.** The Sherali-Adams SDP relaxation is the Sherali-Adams LP relaxation plus the following semi-definiteness constraint. Define M_1 to be an $(n+1) \times (n+1)$ matrix whose rows and columns are indexed by $\emptyset, \{1\}, \ldots, \{n\}$ as follows and add the following semi-definiteness condition.

$$m_{I,J} = \mathop{\mathbb{P}}_{S \sim \mathcal{D}(I \cup J)}[I \cup J \subseteq S] \qquad M_1 = [m_{I,J}]_{(n+1)\times(n+1)} \succeq 0. \tag{3}$$

In other words, one makes a matrix whose first row and column and diagonal are the "singleton probabilities", i.e., the probabilities of each vertex being in a set sampled according

to the local distribution, while the rest of the matrix is filled with the "doubleton probabilities", i.e., the probabilities that pairs of vertices are in a set sampled according to the local distribution together.

It is not hard to see that any *integral* solution of (1) gives rise to a solution to the Sherali-Adams SDP relaxation of any level. It is also not hard to see that the optimum of the Sherali-Adams SDP relaxation can be found in time polynomial in n^t. While the above is not the original definition of Sherali-Adams hierarchy it is equivalent. The reader can see the original definition as well as the formal theorem stating the equivalence in the full version of the paper.

3 Outline of Our Method and Comparison to Previous Work

By Theorem 2.2, for $\gamma = \sqrt{\log m/m}$, G^m_γ has no vertex cover smaller than $2^m(1-o(1))$. A tight integrality gap therefore calls for a solution in the system of objective value at most $2^m(1/2+\epsilon)$, for a small constant $\epsilon > 0$.

Consider the following experiment used to define our solution. A geometric way to obtain a distribution of vertex covers would be to embed G^m_γ on the unit sphere and take a sufficiently large *spherical cap* centered at a random point on the sphere. Of course, given the Frankl-Rödl theorem mentioned above, in doing so we have not achieved much since we are defining a *global* distribution of vertex covers, and thus its expected size has to be at least $2^m(1-o(1))$. However, it is useful to understand why these vertex covers are big from a geometric point of view: the heightof the spherical cap must be at least $1+\sqrt{\gamma}$ (as opposed to 1 for a half-sphere.) Now concentration of measure on the sphere implies that because $\sqrt{\gamma m} = \omega(1)$ the area of such a cap is a $1-o(1)$ fraction of the whole sphere. So the probability that any vertex of the graph is in the cap is $1-o(1)$, which is very large. Had it been the case that $\sqrt{\gamma m} = o(1)$ concentration of measure would imply that the area of the cap is $1/2+o(1)$ of that of the sphere and we would have had a small vertex cover.

The main idea is that one only needs to define probabilities for small sets (up to size t if the goal is to show integrality gaps for level-t Sherali-Adams LP relaxations.) So one can first embed the points in such a small set in a *small dimensional sphere* and then repeat the above experiment to define a random vertex cover. The spherical caps that are required in order to cover the edges in these sets have the same height, but now, due to the lower dimension, their area is greatly reduced! Specifically, if the original set has at most t points, the experiment can be performed in a t-dimensional sphere and if $\sqrt{\gamma t} = o(1)$, the probability of any vertex participating in the vertex cover will be no more than $1/2+o(1)$. In particular, $t = o(\sqrt{m/\log m})$ would suffice.

It is critical, of course, that the obtained distributions are consistent. But this is "built-in" in this experiment. Indeed, due to spherical symmetry, the probability that a set of points on a t dimensional sphere belong to a random cap of a fixed radius depends only on t, the radius of the cap and the pairwise Euclidean distances of the points in the set. Interestingly this construction works for any graph with vector chromatic number $2+o(1)$. In other words, if G is an n vertex graph that can be embedded into the unit sphere so that the end points of any edge are almost antipodes, then there is a sufficiently "low-level" (but non-trivial) Sherali-Adams solution of value $(1/2+o(1))n$.

Unfortunately, we cannot show that the above solution satisfies the extra constraints imposed by the SA SDP system. Instead we change our solution in several ways to attain positive semidefiniteness. These changes are somewhat technical and we avoid discussing them in detail here. At a high level the changes are (i) we add a small probability of

picking the *whole* graph as the vertex cover. (ii) We apply a transformation of the canonical embedding of the cube in the sphere that ensures that the farthest pairs of vertices are precisely the edges, and also that the inner products have a bias to being positive (as opposed to the canonical embedding in which the average inner product is 0.)

To get some insight into the rationale of these modifications, first note that the matrix whose positive definiteness we need to prove happens to be highly symmetric. For such symmetric matrices a necessary condition for positive semi-definiteness is that the average entry is at least as large as the square of the diagonal entries. Manipulation (i) above is precisely the tool we need to ensure this condition, and has no adverse effect otherwise. The second transformation is useful although not clearly necessary. We can, however, argue that without a transformation of this nature, a good SDP solution is possible also for a graphs in which edges connect vertices that are at least as far as $m(1-\gamma)$ (rather than exactly that distance). The existence of solutions for such dense graphs seems intuitively questionable. Last, boosting the typical inner product can be shown to considerably boost the Taylor coefficients of a certain function which we need to show only has positive Taylor coefficients. The later is a condition to which we reduce the positive-semidefiniteness of our LP solution.

Comparison to Previous Work: There are more than half a dozen different integrality gap constructions for Vertex Cover in different Lift-and-Project systems known. Among these the most relevant to our work is [9]. In [9], Charikar, et al. obtain a Sherali-Adams solution that is based on embedding the vertices of the graph in the sphere. The similarity with our work is that Charikar et al. take a special case of caps, i.e. half-spheres, in order to determine probabilities. Consistency of these distribution is, just as in our case, guaranteed by the fact that these probabilities are intrinsic to the local distances of the point-set in question. However, the reason that these distributions behave differently than a global distribution (which is essential for an integrality gap construction) is completely different than ours. It is easy to see that when the caps in the construction are half spheres, the dimension does not play a role at all. However, in [9] there is no *global embedding* of the points in the sphere but rather only a local one. In contrast, our distributions can be defined for all dimensions, however as we mentioned we must keep the dimension reasonably small in order to guarantee small objective value. Another big difference pertains to the different instances. While our construction may very well be the one (or close to the one) that will give a Lasserre integrality-gap bound, the instances of [9] have no substantial integrality gap even for the standard SDP. Thus their result cannot be extended to the stronger Lasserre or Sherali-Adams SDP hierarchies.

It is also important to put our work in context with the sequence of results dealing with SDP integrality gaps of Vertex-Cover [17, 8, 14, 15, 16]. In these works the solution can be thought of as an approximation to a very simple set: a dimension cut, that is a face of the cube. This set is not a vertex cover, but in some geometric sense is close to one. The SDP solutions are essentially averaging of such dimension-cuts with some carefully crafted perturbations. Using the same language, the solution we present is based on Hamming balls of radius $m/2$ (i.e. translations of the majority function) rather than dimension-cuts (i.e. dictatorship functions). The perturbation we apply to make such a solution valid is simply the small increase in the radius of the Hamming balls. Another distinction is that while all previous results use tensoring to *construct* their solutions we mainly use it to *certify* its positive semidefiniteness. In other words, our solutions are defined geometrically and then tensoring is used to give an alternative view which helps to show they have the required positive semidefiniteness.

4 Fooling LPs derived by the Sherali-Adams System

4.1 Local Distributions of Vertex Covers for Borsuk Graphs

In this section we study relaxation (2) for discrete subgraphs of B^m_γ on n vertices. In particular, for every set $U \subseteq [n]$ we define a distribution of vertex covers that are locally consistent.

The family of distributions we are looking for arises from the following experiments. Fix a discrete subgraph $G = (V, E)$ of B^m_δ on n vertices for which we want to construct a level t Sherali-Adams LP solution with small objective value. Given that $G = (V, E)$ is a subgraph of B^m_δ we can think of its vertices as points on S^{m-1} and in particular talk about their Euclidean distances. The following experiment defines the local distributions.

Experiment Local-Global

The input is any $I \subseteq V$, of size at most t, and some $\sqrt{\delta} > 0$.
The result of the experiment is a distribution $\mathcal{D}(I)$ of 0/1 assignments on I.

(a) Embed the I-induced subgraph of G into S^{t-1} preserving all pairwise Euclidean distances.

(b) In S^{t-1} consider the complement C of a random spherical cap of height $1 - \sqrt{\delta}$.

(c) Vertices of I are assigned 1 if they are in the cap C, otherwise they are assigned 0.

Notice that step (a) is possible because $|I| \leq t$.

▶ **Lemma 4.1.** *For every finite subgraph of B^m_δ on n vertices, the family of distributions $\mathcal{D}(I)$, $I \in \mathcal{P}^{[n]}_t$, is a valid solution of (2), i.e. a family of locally consistent distributions of vertex covers.*

Proof. The second constraint of (2), i.e. local consistency, follows from the following simple geometric fact: the probability distribution $\mathcal{D}(I)$ only depends on the pairwise Euclidean distances of vertices in I and the parameter t. Given this simple observation it is not hard to see that $\mathcal{D}(U_1)$ is just the marginal of $\mathcal{D}(U)$ when $U_1 \subseteq U$.

It therefore remains to argue that $\mathcal{D}(I)$ is a distribution of vertex covers, i.e. the first constraint of (2). To that end, we need to show that in the Experiment Local-Global, two adjacent vertices cannot be at the same time outside the random cap C. This is true simply because the cap is big enough. In particular, for any two vertices i, j outside the cap if $\mathbf{z}_i, \mathbf{z}_j$ are their vectors and $\mathbf{w}$ is the vector corresponding to the tip of the cap, $\mathbf{w} \cdot \mathbf{z}_i, \mathbf{w} \cdot \mathbf{z}_j > \sqrt{\delta}$ which implies $\|\mathbf{z}_i + \mathbf{z}_j\| = \|\mathbf{w}\| \, \|(\mathbf{z}_i + \mathbf{z}_j)\| \geq \mathbf{w} \cdot (\mathbf{z}_i + \mathbf{z}_j) > 2\sqrt{\delta}$, where the penult inequality is Cauchy-Schwarz. Since G is a subgraph of B^m_γ, we conclude that ij cannot be en edge. ◀

All that remains is to show that the objective value of (2) for our solution is indeed small. In fact, we can show a stronger statement, not only is the objective value $n/2 + o(n)$ but each vertex roughly contributes $1/2$ to the objective value. In particular we can show the following lemma.

▶ **Lemma 4.2.** *For any fixed $\mathbf{z} \in S^{t-1}$, we have $\mathbb{P}_{\mathbf{w} \in S^{t-1}}[\mathbf{w} \cdot \mathbf{z} \leq \eta] \leq \frac{1}{2} + \eta\sqrt{\frac{\pi}{8}(t+1)}$, when $\mathbf{w}$ is distributed uniformly on S^{t-1}. Consequently, for any vertex $i \in I$ of the graph G (subgraph of B^m_δ), we have $\mathbb{P}_{S \sim \mathcal{D}(I)}[i \in S] \leq \frac{1}{2} + \sqrt{\delta\pi(t+1)/8}$.*

The following theorems follow from Lemma 4.2. The proofs can be found in the full version.

▶ **Theorem 4.3.** *Let G be a finite subgraph of B^m_δ on n vertices. Then the level-$\left(\frac{2\epsilon^2}{\pi}\frac{1}{\delta}-1\right)$ Sherali-Adams relaxation (2) for vertex cover has objective value at most $(1/2+\epsilon)n$ for G.*

▶ **Theorem 4.4.** *For every ϵ, there are graphs on n vertices such that the level-$\Omega(\frac{\log n}{\log\log n})$ LP derived by the Sherali-Adams system for* Vertex Cover *has integrality gap $2-\epsilon$.*

5 Fooling SDPs derived by the Sherali-Adams System

5.1 Preliminary Observations for the Sherali-Adams SDP Solution

Let $\mathbf{y}$ be a Sherali-Adams solution of the LP (2), namely $y_I = \mathbb{P}_{S\sim\mathcal{D}(I)}[I \subseteq S]$. Then $\mathbf{y}$ uniquely determines the matrix $M_1 = M_1(\mathbf{y})$ in (3). In order to establish a Sherali-Adams SDP integrality gap, we need to show that $M_1(\mathbf{y})$ is positive-semidefinite for an appropriately chosen $\mathbf{y}$.

It is convenient to denote by $M'_1(\mathbf{y})$ the principal submatrix $M_1(\mathbf{y})$ indexed by nonempty sets. Note that for the solution we introduced in the previous section, all $y_{\{i\}}$ attain the same value, say y_R. In other words, $M_1(\mathbf{y}) = \begin{pmatrix} 1 & \mathbf{1}y_R \\ \mathbf{1}^T y_R & M'_1(\mathbf{y}) \end{pmatrix}$, where $\mathbf{1}$ denotes the all 1 vector of appropriate size. We leave the proof of the following fact for the full version.

▶ Fact 5.1. Suppose that $\mathbf{1}$ is an eigenvector for $M'_1(\mathbf{y})$. Then $M_1(\mathbf{y}) \succeq 0$ iff $M'_1(\mathbf{y}) \succeq 0$ and for some $j \in V$, $\mathrm{avg}_{i\in V} y_{\{i,j\}} \geq y_R^2$.

The next Lemma establishes a sufficient condition for solutions fooling SDP relaxations for Borsuk graphs. The proof uses the standard tool of tensoring introduced in Section 2.1.

▶ **Lemma 5.2.** *Let $\mathbf{y}$ be a level-t Sherali-Adams solution for* Vertex Cover *for a Borsuk graph with vector representation $\mathbf{u}_i$ and suppose that the value $y_{\{i,j\}}$ can be expressed as $f(\mathbf{u}_i \cdot \mathbf{u}_j)$. If the Taylor expansion of $f(x)$ has no negative coefficients, then $M'_1(\mathbf{y}) \succeq 0$.*

Proof. Consider the Taylor expansion of $f(x) = \sum_{i=0}^{\infty} a_i x^i$, where $a_i \geq 0$. We map $\mathbf{u}_i \in S^{m-1}$ to an infinite dimensional space as follows $\mathbf{u}_i \mapsto T_f(\mathbf{u}_i)$. Then the vectors $T_f(\mathbf{u}_i)$ constitute the Cholesky decomposition of $M'_1(\mathbf{y})$, and therefore $M'_1(\mathbf{y}) \succeq 0$. ◀

Now we examine the Sherali-Adams solution of some special case that will be instructive for our general argument. Consider some n vertex subgraph $G = (V, E)$ of $B^m_{\rho^2}$ with vector representation $\mathbf{z}_i \in S^{m-1}$. Suppose also that edges $ij \in E$ appear exactly when $\mathbf{z}_i \cdot \mathbf{z}_j = -1 + 2\rho^2$, and that for all other pairs $i, j \in V$ we have $\mathbf{z}_i \cdot \mathbf{z}_j \geq -1 + 2\rho^2$. Run Experiment Local-Global with parameters $t = 2$ and $\delta = \rho^2$ to define the level-2 Sherali-Adams solution $\mathbf{y}$

$$y_I = \mathop{\mathbb{P}}_{\mathbf{w}\in S^1}[\mathbf{w}\cdot\mathbf{z}_i \leq \rho,\ \forall i \in I] \tag{4}$$

for all I of size at most 2, where $\mathbf{w}$ is distributed uniformly on the circle.

▶ Claim 5.3. The values $y_{\{i,j\}}$ depend on the inner product $x = \mathbf{z}_i \cdot \mathbf{z}_j$ in the following way: (a) if $2\rho^2 \geq x+1$, $y_{\{i,j\}} = 1 - \frac{2\theta_\rho}{\pi}$, (b) if $2\rho^2 \leq x+1$, $y_{\{i,j\}} = 1 - \frac{\theta_\rho}{\pi}$; where $\theta_x = \arccos(x)$. In particular, when $\mathbf{z}_i \cdot \mathbf{z}_j = 1$, $y_{\{i,j\}} = 1 - \frac{\theta_\rho}{\pi}$.

The next fact is motivated by the condition of Lemma 5.2.

▶ Fact 5.4. If $\rho \in [0,1]$, the Taylor expansion of the function $1 - \frac{\theta_\rho}{\pi} - \frac{\theta_x}{2\pi}$ has no negative coefficient.

We leave the proofs of Claim 5.3 and Fact 5.4 for the full version.

Note that if we start with a configuration of vectors $\mathbf{z}_i$ for which $\mathbf{z}_i \cdot \mathbf{z}_j \geq -1 + 2\rho^2$ for all pairs $i, j \in V$, then the value $y_{\{i,j\}}$ will be described as a function on the inner product $\mathbf{z}_i \cdot \mathbf{z}_j = x$, and this function on x will have Taylor expansion with nonnegative coefficients. Unfortunately, for our Sherali-Adams solution of the previous sections this is not the case. We establish this extra condition in Section 5.2, making sure that $M_1'(\mathbf{y})$ is positive semidefinite. Proving that the matrix $M_1(\mathbf{y})$ is positive semidefinite will require one extra simple argument, which is self evident from fact 5.1.

5.2 An Easy level-2 Sherali-Adams SDP Solution

In this section we apply the techniques developed in Section 5.1 to show a tight integrality gap for Vertex Cover in the level-2 Sherali-Adams SDP system. This serves as an instructive example for higher levels whose proof are a smooth generalization of the arguments below. We will show,

▶ **Theorem 5.5.** *For any $\epsilon > 0$, there exist $\delta > 0$ and sufficiently big m, such that the level-2 Sherali-Adams SDP system for* Vertex Cover *on G_δ^m has objective value at most $2^m(1/2 + \epsilon)$.*

As the theorem states, we start with the Frankl-Rödl graph $G_\delta^m = (V, E)$, which is a subset of B_δ^m, with vector representation $\mathbf{u}_i$. Our goal is to define $\mathbf{y}$ in the context of Theorem 4.3, so as the matrix $M_1(\mathbf{y})$ to be positive semidefinite. Our Sherali-Adams solution as it appears in Theorem 4.3 does not satisfy the constraint $M_1(\mathbf{y}) \succeq 0$, for reasons that will be clear shortly. For this, we need to apply the transformation $\mathbf{u}_i \mapsto \mathbf{z}_i := (\sqrt{\zeta}, \sqrt{1-\zeta}\ T_P(\mathbf{u}_i))$, for some appropriate tensoring polynomial $P(x)$, and some $\zeta > 0$ (that is allowed to be a function of (m, δ)). We will use the following fact, first proved by Charikar [8].

▶ Fact 5.6. There exist a polynomial $P(x)$, with nonnegative coefficients and $P(1) = 1$, such that for all $x \in [-1, 1]$, we have $P(x) \geq P(-1 + 2\delta) = -1 + 2\delta_0$, for some $\delta_0 = \Theta(\delta)$. Moreover, for every constant $c > 0$ and for every $x \in (-c/\sqrt{m}, c/\sqrt{m})$, we have $|P(x)| = O(\sqrt{1/m})$.

We use the polynomial P of Fact 5.6 to map the vectors $\mathbf{u}_i$ to the new vectors $\mathbf{z}_i$. Note that with this transformation, for an edge $ij \in E$ we have $\mathbf{z}_i \cdot \mathbf{z}_j = \zeta + (1-\zeta)P(-1+2\delta) = \zeta + (1-\zeta)(-1+2\delta_0) = -1 + 2\left(\zeta(1-\delta_0) + \delta_0\right)$. If we denote $\sqrt{\zeta(1-\delta_0)+\delta_0}$ by ρ, then the above transformation maps G_δ^m to $G_{\rho^2}^{m'}$, where m' is the degree of the polynomial P. We are therefore eligible to run Experiment Local-Global with parameters $t = 2$ and ρ^2 on the vectors $\mathbf{z}_i = (\sqrt{\zeta}, \sqrt{1-\zeta}\ T_P(\mathbf{u}_i))$. Then Lemma 4.1 implies that $\mathbf{y}$ as defined in (4) is a level-2 Sherali-Adams solution (the parameters δ, ζ will be fixed later). Next we show that for a slightly perturbed $\mathbf{y}$ we have that $M_1(\mathbf{y})$ is positive semidefinite.

First we observe that the context of Section 5.1 is relevant to the current configuration of vectors $\mathbf{z}_i$ and to our graph instances, since $\mathbf{z}_i \cdot \mathbf{z}_j \geq -1 + 2\rho^2$. If $\mathbf{u}_i \cdot \mathbf{u}_j = x$, then the value of $y_{\{i,j\}}$ is exactly $g(\zeta + (1-\zeta)P(x))$, where $g(x) = 1 - \frac{\theta_\rho}{\pi} - \frac{\arccos(x)}{2\pi}$. By Fact 5.4 we know that the function $g(x)$ has Taylor expansion with nonnegative coefficients. Since $\zeta + (1-\zeta)P(x)$ is a polynomial with nonnegative coefficients, it follows that $g(\zeta + (1-\zeta)P(x))$ has Taylor Expansion with nonnegative coefficients. Hence, we can apply Lemma 5.2 to obtain that

▶ **Lemma 5.7.** *The matrix $M_1'(\mathbf{y})$ is positive semidefinite.*

In what follows we describe a way to extend the positive semidefiniteness of $M_1'(\mathbf{y})$ to that of $M_1(\mathbf{y})$. In fact what we will show is general and holds for any level t (where t is the

Sherali-Adams level which solution $\mathbf{y}$ was engineered for). Since the entries of $M_1'(\mathbf{y})$ are a function of the inner product of the corresponding vectors of the hypercube, it follows that the all 1 vector is an eigenvector for $M_1'(\mathbf{y})$. By Fact 5.1 it follows that we need to show that $\text{avg}_{i\in V} y_{\{i,j\}} - y_{\{i\}}^2 \geq 0$. It turns out that this is not the case, but we can establish a weaker condition (described here in terms of a general sphere dimension D).

▶ **Lemma 5.8.** *There exist $c > 0$ (not depending on m, ρ), such that $avg_{i\in V} y_{\{i,j\}} - y_{\{i\}}^2 \geq -cD\rho$.*

We omit the proof of this lemma from this extended abstract. A rough estimate that suffices is that whenever two points have positive inner product, the probability that both are in a random cap is at least $1/4$. It can be shown that due to the affine transformation, all but exponentially small fraction of the pairs will have positive inner products, hence we get that the average of $y_{\{i,j\}}$ is at least $1/4 - o(1)$. On the other hand, from Section 4 we know that $y_{\{i\}} \leq 1/2 + O(D\rho)$.

Boosting: It remains to show how to "boost" the solution to move from the relaxed condition to the exact, and necessary one. The idea is simple. Consider a ridiculously wasteful integral solution to Vertex Cover, namely the solution that takes all vertices. Clearly, if we take a convex combination of this solution with the existing one we still get a Sherali-Adams solution. If the weight of the integral solution is some small number $\xi > 0$ then the objective value increases by no more than $\xi/2$ which can be absorbed for arguments to go through as long as $\xi \leq \epsilon$. Owing to the strict convexity of the quadratic function, however, this simple perturbation does allow to improve the bound on averages as required by Fact 5.1. This observation is made precise in the following Lemma whose proof can be found in the full version.

▶ **Lemma 5.9.** *Let y' be the matrix $y' = (1-\xi)y + \xi J$ where J represents the all 1 solution. Also let $s = y_{\{i\}}$ and $s' = y'_{\{i\}}\}$. Then $avg_{i,j} y'_{\{i,j\}} - s'^2 = \Omega(\xi)$.*

We are now ready to formally prove Theorem 5.5.

Proof. (of Theorem 5.5) We start with the n-vertex Frankl-Rödl graph G_δ^m, with $\delta = \Theta(\frac{\log n}{\log\log n})$ so as to satisfy the conditions of Theorem 2.2. We use the polynomial of Fact 5.6 to obtain the vectors $\mathbf{z}_i = (\sqrt{\zeta}, \sqrt{1-\zeta}\ T_P(\mathbf{u}_i))$, with $\zeta = \delta_0$ (where $\delta_0 = \Theta(\delta)$ by Fact 5.6). We set $\rho = \sqrt{\zeta(1-\delta_0)+\delta_0} = \sqrt{\Theta(\zeta)}$, and we run the Experiment Local-Global on the vectors $\mathbf{z}_i$ with parameters $t = 2$ and ρ^2, to obtain the vector $\mathbf{y}$. By Lemma 4.1, we have that $\mathbf{y}$ as defined in (4) is a level-2 Sherali-Adams solution. Note that since $\delta = o(1)$ we conclude from Lemma 4.2 that $y_{\{i\}} = 1/2 + \Theta(\delta)$.

Next we define $\mathbf{y}'$ as $(1-\xi)y + \xi J$. We already argued that $M_1'(\mathbf{y}')$ is positive semidefinite. By the above discussion (and Lemma 5.9) we conclude that $\text{avg}_{i,j} y'_{\{i,j\}} - y'^2_{\{i\}} \geq 0$. We can therefore use Fact 5.1 to conclude that $M(\mathbf{y}') \succeq 0$. The last thing to note is that the contribution of every vertex in the objective value is $1/2 + O(\delta)$ ◀

5.3 The Level-$(t+2)$ Sherali-Adams SDP Tight Integrality Gap

For the level-$(t+2)$ SDP, we start with the n-vertex Frankl-Rödl graphs G_δ^m, $n = 2^m$ with vector representation $\mathbf{u}_i$. The value of δ is chosen so as to satisfy Theorem 2.2, namely $\delta = \Theta(\sqrt{\log m/m})$. As in Section 5.2 we apply to $\mathbf{u}_i$ two transformations; one using the tensoring polynomial of Fact 5.6 and one affine transformation. Then we use the resulting vectors $\mathbf{z}_i = (\sqrt{\zeta}, \sqrt{1-\zeta}\ T_P(\mathbf{u}_i))$ to define a level-$(t+2)$ Sherali-Adams solution that we denote by $\mathbf{y}$. Our construction of $\mathbf{y}$ will have a parameter ρ to be set later.

Our goal is to meet the conditions of Fact 5.1. Namely, the first thing to ensure is that $M_1'(\mathbf{y})$ is positive semidefinite. In this direction, from Lemma 5.2 it suffices to show that the Taylor expansion of the function that describes the value of $y_{\{i,j\}}$, when $\mathbf{u}_i \cdot \mathbf{u}_j = u$, has Taylor expansion with nonnegative coefficients. Given that this function at 0 will always represent some probability, the problem is equivalent to showing that the first derivative of this function has such a good Taylor expansion. Our transformation on the vectors $\mathbf{u}_i$ can be thought as mapping their inner product u first to $x = P(u)$, and second x to $\kappa_\zeta(x) = \zeta + (1-\zeta)x$. Under this notation, we can show the following lemma that involves a number of technical calculations. The proof can be found in the full version of the paper.

▶ **Lemma 5.10.** *The derivative of the functional description of $y_{\{i,j\}}$ is*

$$D_\zeta(x) := -(\arccos(\kappa_\zeta(x)))'(1 - \frac{2\rho^2}{1+\kappa_\zeta(x)})^{t/2}.$$

Therefore, to conclude that $M_1'(\mathbf{y}) \succeq 0$ it suffices to show the next technical lemma. The proof requires arguments along the lines of that of Claim 5.7 and will appear in the full version.

▶ **Lemma 5.11.** *Set $t = 4$ and $\rho^2 \in [\zeta, \zeta + \zeta^3]$. Then for sufficiently small ζ, the function $D_\zeta(x)$ as it reads in Lemma 5.10 has Taylor expansion with nonnegative coefficients.*

Now we are ready to prove Theorem 1.1. First we obtain a level-$(t+2)$ Sherali-Adams solution from the vectors $\mathbf{z}_i = (\sqrt{\zeta}, \sqrt{1-\zeta}\, T_P(\mathbf{u}_i))$ (the reader may think of $t = 4$). We need to set $\zeta = \sqrt[3]{\delta_0}$, where $\delta_0 = (1 + \min(P(x))/2$. Since the rounding parameter we need is $\rho = \sqrt{\zeta(1-\delta_0) + \delta_0}$, it is easy to see that $\rho^2 = \zeta + \zeta^3 - \zeta^4$. It follows by Lemma 5.11 that the matrix $M_1'(\mathbf{y})$ is positive semidefinite.

Now call c the constant for which $\text{avg}_{i\in V} y_{\{i,j\}} - y_{\{i\}}^2 \geq -ct\rho^2$. We also know that if $t\rho^2$ is no more than a small constant $\epsilon/10$, then $y_{\{i\}} \leq 1/2 + \epsilon$. Then define $\mathbf{y}' = (1 - 4c\epsilon)\mathbf{y} + (4c\epsilon)\mathbf{1}$. As we did for the level-2 Sherali-Adams SDP solution, the vector $\mathbf{y}'$ is a level-$(t+2)$ Sherali-Adams solution. Moreover, the matrix $M_1'(\mathbf{y}')$ is positive semidefinite, and $\text{avg}_{i\in V} y'_{\{i,j\}} - {y'}^2_{\{i\}} \geq 0$. All conditions of Fact 5.1 are satisfied implying that $M_1(\mathbf{y}')$ is positive semidefinite. Finally, note that the contribution of the singletons is no more than $1/2 + \Theta(ct\rho^2)$. Hence, if we start with $t\rho^2 = o(1)$, the contribution of the singletons remains $1/2 + o(1)$. On the other hand, choosing $\delta = \Theta(\sqrt{\log m/m})$ results in graphs G_δ^m with no vertex cover smaller than $n - o(n)$.

The maximum value of t in Lemma 5.11 dictates the limitation on the level of our integrality gap. In particular we have the following conjecture and the proof of Theorem 1.2 is straightforward.

▶ Conjecture 5.12. Set t be any even integer and $\rho^2 \in [\zeta, \zeta + \zeta^3]$. Then for sufficiently small ζ, the function $D_\zeta(x)$ as it reads in Lemma 5.10 has Taylor expansion with nonnegative coefficients.

▶ **Theorem 5.13.** *Assuming Conjecture 5.12, for every constants $\epsilon > 0$ and t, the level-t SDP derived by the Sherali-Adams SDP system for* Vertex Cover *has integrality gap $2 - \epsilon$.*

▶ Remark. [**On the validity of Conjecture 5.12**] Evidence for the validity of Conjecture 5.12 is both experimental and theoretical. In particular, some relatively simple arguments can show the following two statements: (a) For every $N_0 > 0$ there exist small enough $\zeta > 0$, such that the first N_0 Taylor coefficients of $D_\zeta(x)$ are positive, (b) For every $\zeta > 0$, there exist $N_0 > 0$ such that all *but* the first N_0 Taylor coefficients of $D_\zeta(x)$ are positive. While these partial results are not enough to imply Sherali-Adams SDP lowerbounds, they do seem to indicate that Conjecture 5.12 is true.

Discussion

We presented tight integrality gaps for level-6 Sherali-Adams SDP for Vertex Cover and how if a certain analytical conjecture is proved they can be extended to any constant number of rounds. Along the way we also gave an intuitive and geometric proof of tight Sherali-Adams LP integrality gaps for the same problem. While these LP integrality gaps apply to less rounds than [9] they remain highly nontrivial, yet significantly simplified.

For large t, proving Conjecture 5.12 seems challenging. We leave it as an open problem. Another open problem is to extend the ideas in this paper to construct tight Lasserre gaps for Vertex Cover and Unique Games, thus giving the strongest evidence that Unique Games cannot be solved with SDP hierarchies.

Acknowledgements: The authors wish to thank Toniann Pitassi for many helpful discussions and comments on an earlier version of the paper. The authors are also grateful to the anonymous reviewers for suggestions on how to improve the presentation.

References

1 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for Unique Games and related problems. In *FOCS'10*. IEEE Computer Society, 2010.

2 Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):1–37, 2009.

3 Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *FOCS'11*, 2011. To appear.

4 Mohammad Hossein Bateni, Moses Charikar, and Venkatesan Guruswami. MaxMin allocation via degree lower-bounded arborescences. In *STOC'09*, pages 543–552. ACM Press, 2009.

5 Siavosh Benabbas, Siu On Chan, Konstantinos Georgiou, and Avner Magen. The Sherali-Adams system applied to Vertex Cover: Why Borsuk graphs fool strong LPs and some tight Integrality Gaps for SDPs. *ECCC*, 17:169, 2011. Revision 2.

6 Siavosh Benabbas, Konstantinos Georgiou, Avner Magen, and Madhur Tulsiani. SDP gaps from pairwise independence. http://www.cs.toronto.edu/∼siavosh/pdf/pairwise-full-1.pdf, 2010.

7 Siavosh Benabbas and Avner Magen. Extending SDP integrality gaps to Sherali-Adams with applications to Quadratic Programming and MaxCutGain. In *IPCO'10*, pages 299–312. Springer, 2010.

8 Moses Charikar. On semidefinite programming relaxations for graph coloring and Vertex Cover. In *SODA'02*, pages 616–620. ACM Press, 2002.

9 Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Integrality gaps for Sherali-Adams relaxations. In *STOC'09*, pages 283–292. ACM Press, 2009.

10 Eden Chlamtac and Gyanit Singh. Improved approximation guarantees through higher levels of SDP hierarchies. In *APPROX'08*, pages 49–62. Springer-Verlag, 2008.

11 Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of Max Cut. In *SODA'07*, pages 53–61. ACM Press, 2007.

12 Irit Dinur and Shmuel Safra. On the hardness of approximating minimum Vertex Cover. *Annals of Mathematics*, 162(1):439–486, 2005.

13 Peter Frankl and Vojtech Rödl. Forbidden intersections. *Trans. Amer. Math. Soc.*, 300(1):259–286, 1987.

14 Konstantinos Georgiou, Avner Magen, Toniann Pitassi, and Iannis Tourlakis. Integrality gaps of $2 - o(1)$ for Vertex Cover SDPs in the Lovász-Schrijver hierarchy. In *FOCS'07*, pages 702–712. IEEE Computer Society, 2007.

15 Konstantinos Georgiou, Avner Magen, and Iannis Tourlakis. Vertex Cover resists SDPs tightened by local hypermetric inequalities. In *IPCO'08*, pages 140–153. Springer, 2008.

16 Konstantinos Georgiou, Avner Magen, and Iannis Tourlakis. On the tightening of the standard SDP for Vertex Cover with ℓ_1 inequalities. In *FSTTCS'09*, pages 203–214. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.

17 Michel X. Goemans and Jon Kleinberg. The Lovász theta function and a semidefinite programming relaxation of Vertex Cover. *SIAM J. Discrete Math.*, 11(2):196–204 (electronic), 1998.

18 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for Maximum Cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

19 Venkatesan Guruswami and Ali Kemal Sinop. Lasserre hierarchy, higher eigenvalues, and approximation schemes for quadratic integer programming with PSD objectives. *CoRR*, abs/1104.4746, 2011.

20 Hamed Hatami, Avner Magen, and Evangelos Markakis. Integrality gaps of semidefinite programs for Vertex Cover and relations to ℓ_1 embeddability of negative type metrics. *SIAM J. Discret. Math.*, 23:178–194, December 2008.

21 George Karakostas. A better approximation ratio for the Vertex Cover problem. *ACM Trans. Algorithms*, 5(4):1–8, 2009.

22 Anna R. Karlin, Claire Mathieu, and C. Thach Nguyen. Integrality gaps of linear and semi-definite programming relaxations for knapsack. In *IPCO'11*, pages 301–314. Springer-Verlag, 2011.

23 Howard Karloff and Uri Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *FOCS'07*, pages 406–415. IEEE Computer Society, 1997.

24 Subhash Khot. On the power of unique 2-prover 1-round games. In *STOC'02*, pages 767–775. ACM Press, 2002.

25 Subhash Khot. On the Unique Games Conjecture (Invited Survey). In *CCC'10*, pages 99–121. IEEE Computer Society, 2010.

26 Subhash Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1. In *FOCS'05*, pages 53–62. IEEE Computer Society, 2005.

27 Avner Magen and Mohammad Moharrami. Robust algorithms for maximum independent set on minor-free graphs based on the Sherali-Adams hierarchy. In *APPROX'09*, pages 258–271. Springer, 2009.

28 Prasad Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *STOC'08*, pages 245–254. ACM Press, 2008.

29 Prasad Raghavendra and David Steurer. Integrality gaps for strong SDP relaxations of Unique Games. In *FOCS'09*, pages 575–585. IEEE Computer Society, 2009.

30 Grant Schoenebeck. Linear level Lasserre lower bounds for certain k-CSPs. In *FOCS'08*, pages 593–602. IEEE Computer Society, 2008.

31 Grant Schoenebeck, Luca Trevisan, and Madhur Tulsiani. A linear round lower bound for Lovász-Schrijver SDP relaxations of Vertex Cover. In *CCC'07*, pages 205–216. IEEE Computer Society, 2007.

32 Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discrete Math.*, 3(3):411–430, 1990.

33 Madhur Tulsiani. CSP gaps and reductions in the Lasserre hierarchy. In *STOC'09*, pages 303–312. ACM Press, 2009.

34 Sundar Vishwanathan. On hard instances of approximate Vertex Cover. *ACM Trans. Algorithms*, 5(1):Art. 7, 6, 2009.

Applications of Discrepancy Theory in Multiobjective Approximation

Christian Glaßer, Christian Reitwießner, and Maximilian Witek

Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Theoretische Informatik
Am Hubland, 97074 Würzburg, Germany
{glasser,reitwiessner,witek}@informatik.uni-wuerzburg.de

Abstract

We apply a multi-color extension of the Beck-Fiala theorem to show that the multiobjective maximum traveling salesman problem is randomized $1/2$-approximable on directed graphs and randomized $2/3$-approximable on undirected graphs. Using the same technique we show that the multiobjective maximum satisfiabilty problem is $1/2$-approximable.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Discrepancy Theory, Multiobjective Optimization, Satisfiability, Traveling Salesman

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.55

1 Introduction

We study multiobjective variants of the traveling salesman problem and the satisfiability problem.

- The k-objective maximum traveling salesman problem: Given is a directed / undirected complete graph with edge weights from $\mathbb{N}^k$. Find a Hamiltonian cycle of maximum weight.
- The k-objective maximum weighted satisfiability problem: Given is a Boolean formula in conjunctive normal form and for each clause a non-negative weight in $\mathbb{N}^k$. Find a truth assignment that maximizes the sum of the weights of all satisfied clauses.

In general we cannot expect to find a single solution that is optimal with respect to all objectives. Instead we are interested in the *Pareto set* which consists of all optimal solutions in the sense that there is no solution that is at least as good in all objectives and better in at least one objective. Typically, the Pareto set has exponential size, and this particularly holds for the traveling salesman and the satisfiability problems considered here. We are hence interested in computing an approximation of the Pareto set.

A popular strategy for approximating single-objective traveling salesman and single-objective satisfiability is to compute two or more alternatives out of which one chooses the best one:

- For each cycle in a maximum cycle cover of a graph, remove the edge with *the lowest weight*, and connect the remaining paths to a Hamiltonian cycle.
- For some formula, take an arbitrary truth assignment and its complementary truth assignment, and return the one with *the highest weight* of satisfied clauses.

However, in the presence of multiple objectives, these alternatives can be incomparable and hence we need an argument that allows to *appropriately combine* incomparable alternatives.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 55–65

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While previous work focused on problem-specific properties to construct solutions of good quality, we show that the Beck-Fiala theorem [3] from discrepancy theory and its multi-color extension due to Doerr and Srivastav [5] provide a general and simple way to combine alternatives appropriately. Its application leads to simplified and improved approximation algorithms for the k-objective maximum traveling salesman problem on directed and undirected graphs and the k-objective maximum weighted satisfiability problem.

2 Preliminaries

2.1 Multiobjective Optimization

Let $k \geq 1$ and consider some k-objective maximization problem $\mathcal{O}$ that consists of a set of instances $\mathcal{I}$, a set of solutions $S(x)$ for each instance $x \in \mathcal{I}$, and a function w assigning a k-dimensional weight $w(x,s) \in \mathbb{N}^k$ to each solution $s \in S(x)$ depending also on the instance $x \in \mathcal{I}$. If the instance x is clear from the context, we also write $w(s) = w(x,s)$. The components of w are written as w_i. For weights $a = (a_1, \ldots, a_k)$, $b = (b_1, \ldots, b_k) \in \mathbb{N}^k$ we write $a \geq b$ if $a_i \geq b_i$ for all i.

Let $x \in \mathcal{I}$. The Pareto set of x, the set of all optimal solutions, is the set $\{s \in S(x) \mid \neg\exists s' \in S(x)\ (w(x,s') \geq w(x,s) \text{ and } w(x,s') \neq w(x,s))\}$. For solutions $s, s' \in S(x)$ and $\alpha < 1$ we say s is α-approximated by s' if $w_i(s') \geq \alpha \cdot w_i(s)$ for all i. We call a set of solutions α-approximate Pareto set of x if every solution $s \in S(x)$ (or equivalently, every solution from the Pareto set) is α-approximated by some s' contained in the set.

We say that some algorithm is an α-approximation algorithm for $\mathcal{O}$ if it runs in polynomial time and returns an α-approximate Pareto set of x for all inputs $x \in \mathcal{I}$. We call it randomized if it is allowed to fail with probability at most $1/2$. An algorithm is a PTAS (polynomial-time approximation scheme) for $\mathcal{O}$, if on input x and $0 < \varepsilon < 1$ it computes a $(1-\varepsilon)$-approximate Pareto set of x and for each fixed ε, its runtime is polynomial in the length of x. If there is a single polynomial in $1/\varepsilon + \text{length}(x)$ that bounds the algorithm's runtime, we call it FPTAS (fully polynomial-time approximation scheme). The randomized variants are called PRAS (polynomial-time randomized approximation scheme) and FPRAS (fully polynomial-time randomized approximation scheme).

2.2 Graph Prerequisites

An *$\mathbb{N}^k$-labeled directed (undirected) graph* is a tuple $G = (V, E, w)$, where V is some finite set of vertices, $E \subseteq V \times V$ ($E \subseteq \binom{V}{2}$) is a set of directed (undirected) edges, and $w\colon E \to \mathbb{N}^k$ is a k-dimensional weight function. If $E = (V \times V) \setminus \{(i,i) \mid i \in V\}$ ($E = \binom{V}{2}$) then G is called *complete*. We denote the i-th component of w by w_i and extend w to sets of edges by taking the sum over the weights of all edges in the set. A *cycle (of length $m \geq 1$)* in G is an alternating sequence of vertices and edges $v_0, e_1, v_1, \ldots e_m, v_m$, where $v_i \in V$, $e_j \in E$, $e_j = (v_{j-1}, v_j)$ ($e_j = \{v_{j-1}, v_j\}$) for all $0 \leq i \leq m$ and $1 \leq j \leq m$, neither the sequence of vertices $v_0, v_1, \ldots, v_{m-1}$ nor the sequence of edges $e_1, \ldots, e_m$ contains any repetition, and $v_m = v_0$. A cycle in G is called *Hamiltonian* if it visits every vertex in G. A set of cycles in G is called *cycle cover* if for every vertex $v \in V$ it contains exactly one cycle that visits v. For simplicity we interpret cycles and cycle covers as sets of edges and can thus (using the above mentioned extension of w to sets of edges) write $w(C)$ for the (multidimensional) weight of a cycle cover C of G.

2.3 Approximating Cycle Covers

We will consider approximation algorithms for the multiobjective traveling salesman problem using a multiobjective version of the maximum cycle cover problem. For directed input graphs we have the following problem definition.

k-Objective Maximum Directed Edge-Fixed c-Cycle Cover (k-c-MaxDCC$_{\mathrm{F}}$)
Instance: $\mathbb{N}^k$-labeled complete directed graph (V, E, w) and $F \subseteq E$
Solution: Cycle cover $C \subseteq E$ with at least c edges per cycle and $F \subseteq C$
Weight: $w(C)$

For undirected input graphs we analogously define the k-objective maximum undirected edge-fixed c-cycle cover problem (***k*-*c*-MaxUCC$_{\mathbf{F}}$**, for short). Let ***k*-*c*-MaxUCC** (***k*-*c*-MaxDCC**) denote the problems we obtain from k-c-MaxDCC$_{\mathrm{F}}$ (k-c-MaxUCC$_{\mathrm{F}}$) if we require $F = \emptyset$. Using this notation we obtain the usual cycle cover problems ***k*-MaxDCC** as k-0-MaxDCC and ***k*-MaxUCC** as k-0-MaxUCC.

Manthey and Ram [14] show by a reduction to matching that there is an FPRAS for k-objective *minimum* cycle cover problems. The same technique can be used to show that there are FPRAS for k-MaxDCC and k-MaxUCC [12]. We show that there are FPRAS for k-2-MaxDCC$_{\mathrm{F}}$ and k-3-MaxUCC$_{\mathrm{F}}$ by a reduction to k-MaxDCC and k-MaxUCC.

▶ **Theorem 1.** *For every $k \geq 1$, k-2-MaxDCC$_{\mathrm{F}}$ and k-3-MaxUCC$_{\mathrm{F}}$ admit an FPRAS.*

Proof. For every $l \geq 1$, let l-MaxDCC-Approx (l-MaxUCC-Approx) denote the FPRAS for l-MaxDCC (l-MaxUCC). We begin with the directed case.

Let $k \geq 1$. On input of the $\mathbb{N}^k$-labeled complete directed graph $G = (V, E, w)$ and $F \subseteq E$, let $G' = (V, E, w')$, where $w' \colon E \to \mathbb{N}^{k+1}$ such that for all $e \in E$,

$$w'_i(e) = w_i(e) \qquad \text{for } 1 \leq i \leq k \qquad \text{and} \qquad w'_{k+1}(e) = \begin{cases} 1 & \text{if } e \in F \\ 0 & \text{otherwise.} \end{cases}$$

For $\varepsilon > 0$, apply $(k+1)$-MaxDCC-Approx to G' with approximation ratio $\varepsilon' = \min\{\varepsilon, 1/(r+1)\}$, where $r := \#F$ and return the obtained set of cycle covers that contain all edges from F.

Let C be some (arbitrary) cycle cover with $F \subseteq C$. If no such cycle cover exists, we are done. Otherwise, we have $w'_{k+1}(C) = r$, and with probability at least $1/2$ the FPRAS must have returned some cycle cover C' that ε'-approximates C. By $\varepsilon' \leq 1/(r+1)$ we have $w'_{k+1}(C') \geq (1 - \varepsilon') \cdot w'_{k+1}(C) \geq (1 - 1/(r+1)) \cdot r = r - r/(r+1) > r - 1$ and hence $F \subseteq C'$. Moreover, by $\varepsilon' \leq \varepsilon$ we have $w_i(C') = w'_i(C') \geq (1-\varepsilon') \cdot w'_i(C) \geq (1-\varepsilon) \cdot w'_i(C) = (1-\varepsilon) \cdot w_i(C)$ for all $1 \leq i \leq k$. Since an arbitrary cycle in a complete directed graph has length at least two, the assertion is proved.

The proof for the undirected case is very similar, as we call $(k+1)$-MaxUCC-Approx instead. Since in a complete undirected graph every cycle has length at least three, the assertion follows. ◀

2.4 Boolean Formulas

We consider formulas over a finite set of propositional variables V, where a *literal* is a propositional variable $v \in V$ or its negation $\overline{v}$, a *clause* is a finite, nonempty set of literals, and a *formula in conjunctive normal form* (**CNF**, for short) is a finite set of clauses. A *truth assignment* is a mapping $I \colon V \to \{0, 1\}$. For some $v \in V$, we say that I *satisfies the literal* v if $I(v) = 1$, and I *satisfies the literal* $\overline{v}$ if $I(v) = 0$. We further say that I *satisfies the clause* C and write $I(C) = 1$ if there is some literal $l \in C$ that is satisfied by I.

3 Multi-Color Discrepancy

Suppose we have a list of items with (single-objective) weights and want to find a subset of these items with about half of the total weight. The exact version of this problem is of course the NP-complete problem Partition [7], and hence it is unlikely that an exact solution can be found in polynomial time. If we allow a deviation in the order of the largest weight, this problem can be solved in polynomial time, though. Surprisingly, this is still true if the weights are not single numbers but vectors of numbers, which follows from a classical result in discrepancy theory known as the Beck-Fiala theorem [3]. It is important to note that the allowed deviation is independent of the number of vectors since this enables us to use this result in multiobjective approximation for balancing out multiple objectives at the same time with an error that does not depend on the input size.

In the Beck-Fiala theorem and the task discussed above, we have to decide for each item to either include it or not. In some situations in multiobjective optimization, though, a more general problem needs to be solved: There is a constant number of weight vectors for each item, out of which we have to choose exactly one. Doerr and Srivastav [5] showed that the Beck-Fiala theorem generalizes to this so-called multi-color setting with almost the same deviation. Their proof implicitly shows that this choice can be computed in polynomial time.

For a vector $x \in \mathbb{Q}^m$ let $||x||_\infty = \max_i |x_i|$, and for a matrix $A \in \mathbb{Q}^{m\times n}$ let $||A||_1 = \max_j \sum_i |a_{ij}|$. For $c \geq 2$, $n \geq 1$ let $\overline{M_{c,n}} = \{x \in (\mathbb{Q} \cap [0,1])^{cn} \mid \sum_{k=0}^{c-1} x_{cb-k} = 1 \text{ for all } b \in \{1,\dots,n\}\}$ and $M_{c,n} = \overline{M_{c,n}} \cap \{0,1\}^{cn}$.

▶ **Theorem 2.** *(Doerr, Srivastav [5]) There is a polynomial-time algorithm that on input of some $A \in \mathbb{Q}^{m\times cn}$, $m,n \in \mathbb{N}$, $c \geq 2$ and $p \in \overline{M_{c,n}}$ finds a coloring $\chi \in M_{c,n}$ such that $||A(p-\chi)||_\infty \leq 2||A||_1$.*

▶ **Corollary 3.** *There is a polynomial-time algorithm that on input of a set of vectors $v^{j,r} \in \mathbb{Q}^m$ for $1 \leq j \leq n$, $1 \leq r \leq c$ computes a coloring $\chi\colon \{1,\dots,n\} \to \{1,\dots,c\}$ such that for each $1 \leq i \leq m$ it holds that*

$$\left| \frac{1}{c} \sum_{j=1}^{n} \sum_{r=1}^{c} v_i^{j,r} - \sum_{j=1}^{n} v_i^{j,\chi(j)} \right| \leq 2m \max_{j,r} |v_i^{j,r}|.$$

Proof. The result is obvious for $c = 1$. For $c \geq 2$, we use Theorem 2. Because the error bound is different for each row, we need to scale the rows of the vectors. Let $\delta_i = \max_{j,r} |v_i^{j,r}|$ for $1 \leq i \leq m$. Let $A = (a_{i,j'}) \in \mathbb{Q}^{m\times cn}$ where $a_{i,(c(j-1)+r)} = \frac{1}{\delta_i} v_i^{j,r}$ (if $\delta_i = 0$, set it to 0) and $p \in \mathbb{Q}^{cn}$ such that $p_i = \frac{1}{c}$ for all $1 \leq i \leq cn$. We obtain a coloring $\chi \in \{0,1\}^{cn}$ such that for each $1 \leq j \leq n$ there is exactly one $1 \leq r \leq c$ such that $\chi_{c(j-1)+r} = 1$ and it holds that $||A(p-\chi)||_\infty \leq 2||A||_1$. Note that because of the scaling, the largest entry in A is 1 and thus we have $||A||_1 \leq m$. Define $\chi'\colon \{1,\dots,n\} \to \{1,\dots,c\}$ by $\chi'(j) = r \iff \chi_{c(j-1)+r} = 1$. For each $1 \leq i \leq m$ we obtain

$$\begin{aligned} 2m\delta_i &\geq 2||\delta_i\, A||_1 \geq |(\delta_i\, A(p-\chi))_i| \\ &= \left| \sum_{j'=1}^{cn} \delta_i a_{ij'} (p_{j'} - \chi_{j'}) \right| = \left| \sum_{j=1}^{n} \sum_{r=1}^{c} \frac{1}{c} v_i^{j,r} - \sum_{j=1}^{n} v_i^{j,\chi'(j)} \right|. \end{aligned}$$

◀

4 Approximating Multiobjective Maximum Traveling Salesman

4.1 Definition

Given some complete $\mathbb{N}^k$-labeled graph as input, our goal is to find a Hamiltonian cycle of maximum weight. For directed graphs this problem is called k-objective maximum asymmetric traveling salesman (k-MaxATSP), while for undirected graphs it is called k-objective maximum symmetric traveling salesman (k-MaxSTSP). Below we give the formal definition of k-MaxATSP, the problem k-MaxSTSP is defined analogously.

k-Objective Maximum Asymmetric Traveling Salesman (k-MaxATSP)
Instance: $\mathbb{N}^k$-labeled directed complete graph (V, E, w)
Solution: Hamiltonian cycle C
Weight: $w(C)$

4.2 Previous Work

In 1979, Fisher, Nemhauser and Wolsey [6] gave a $1/2$-approximation algorithm for single-objective maximum asymmetric traveling salesman (1-MaxATSP) by removing the lightest edge from each cycle of a maximum cycle cover and connecting the remaining paths to a Hamiltonian cycle. Since undirected cycles always contain at least three edges, this also showed that single-objective maximum symmetric traveling salesman (1-MaxSTSP) is $2/3$-approximable. Since then, many improvements were achieved, and currently, the best known approximation ratios of $2/3$ for 1-MaxATSP and $7/9$ for 1-MaxSTSP are due to Kaplan et al. [9] and Paluch, Mucha and Madry [15].

Most single-objective approximation algorithms do not directly translate to the case of multiple objectives, and hence we need more sophisticated algorithms. For k-MaxATSP and k-MaxSTSP, where $k \geq 2$, the currently best known approximation algorithms are due to Manthey, who showed a randomized $(1/2-\varepsilon)$-approximation of k-MaxATSP and a randomized $(2/3-\varepsilon)$-approximation of k-MaxSTSP [12]. Recently, Manthey also gave a deterministic $(1/2k-\varepsilon)$-approximation of k-MaxSTSP and a deterministic $(1/(4k-2)-\varepsilon)$-approximation of k-MaxATSP [13].

4.3 Our Results

We show that k-MaxATSP is randomized $1/2$-approximable and k-MaxSTSP is randomized $2/3$-approximable using the following idea. We choose a suitable number l depending only on k and try all sets of at most l edges F using brute force. For each such F we apply the FPRAS for k-2-MaxDCC$_\mathrm{F}$ (k-3-MaxUCC$_\mathrm{F}$), which exists by Theorem 1, fixing the edges in F. For all cycle covers thus obtained, we select two (three) edges from each cycle and compute a 2-coloring (3-coloring) of the cycles with low discrepancy with regard to the weight vectors of the selected edges. Using this coloring, we remove exactly one edge from each cycle and connect the remaining simple paths to a single cycle in an arbitrary way. Since the coloring has low discrepancy, we only remove about one half (one third) of the weight in each objective. The introduced error is absorbed by choosing suitable heavy edges F at the beginning. The described procedure generally works for arbitrary c-cycle covers.

▶ **Lemma 4.** *Let $c \geq 2$ and $k \geq 1$. If there exists an FPRAS for k-c-MaxDCC$_\mathrm{F}$ (k-c-MaxUCC$_\mathrm{F}$, resp.), then the algorithm* Alg-k-MaxTSP *computes a randomized $(1-1/c)$-approximation for k-MaxATSP (k-MaxSTSP, resp.).*

Algorithm: Alg-k-MaxTSP(V, E, w) with parameter $c \geq 2$

Input : $\mathbb{N}^k$-labeled directed/undirected complete graph $G = (V, E, w)$
Output : set of Hamiltonian cycles of G

1 `foreach` $F_H, F_L \subseteq E$ with $\#F_H \leq 3\,c\,k^2$, $\#F_L \leq c\#F_H$ `do`
2 let $\delta \in \mathbb{N}^k$ with $\delta_i = \max\{n \in \mathbb{N} \mid$ there are $3\,c\,k$ edges $e \in F_H$ with $w_i(e) \geq n\}$;
3 `foreach` $e \in E \setminus F_H$ `do`
4 `if` $w(e) \not\leq \delta$ `then` set $w(e) = 0$ for the current iteration of line 1;
5 compute $(1 - 1/\#V)$-approx. $\mathcal{S}$ of k-c-MaxDCC$_{\mathrm{F}}$ / k-c-MaxUCC$_{\mathrm{F}}$ on $(G, F_H \cup F_L)$;
6 `foreach` cycle cover $S \in \mathcal{S}$ `do`
7 let $C_1, \ldots, C_r$ denote the cycles in S;
8 `if` for each $i \in \{1, \ldots, r\}$, $C_i \setminus F_H$ contains a path of length c `then`
9 `foreach` $i \in \{1, \ldots, r\}$ `do` choose path $e_{i,1}, \ldots, e_{i,c} \in C_i \setminus F_H$ arbitrarily;
10 compute some coloring $\chi\colon \{1, \ldots, r\} \to \{1, \ldots, c\}$ such that

$$\sum_{i=1}^{r} w(e_{i,\chi(i)}) \quad \leq \quad 2k \cdot \delta + \frac{1}{c} \sum_{i=1}^{r} \sum_{j=1}^{c} w(e_{i,j})$$

  and remove the edges $\{e_{i,\chi(i)} \mid 1 \leq i \leq r\}$ from S;
11 output the remaining edges, arbitrarily connected to a Hamiltonian cycle;

Proof. Let $k \geq 1$, $c \geq 2$, and $G = (V, E, w)$ be some $\mathbb{N}^k$-labeled (directed or undirected) input graph with $m = \#V$ sufficiently large.

We will first argue that the algorithm terminates in time polynomial in the length of G. Since there are only polynomially many subsets $F_H, F_L \subseteq E$ with cardinality bounded by a constant, the loop in line 1 is executed polynomially often. In each iteration the FPRAS on $G = (V, E, w)$ and $F_H \cup F_L \subseteq E$ terminates in time polynomial in the length of G and $F_H \cup F_E$, which means that the set $\mathcal{S}$ contains only polynomially many cycle covers. Hence, for each iteration of the loop in line 1, the loop in line 6 is also executed at most polynomially many times, and overall we have polynomially many nested iterations. In each nested iteration where each cycle of the cycle cover contains a path as required, we compute a coloring of $\{1, \ldots, r\}$ with low discrepancy. By Corollary 3 this can be done in polynomial time. Observe that all further steps require at most polynomial time, and hence the algorithm terminates after polynomially many steps.

Next we argue that the algorithm will succeed with probability at least $1/2$. Observe that the only randomized parts of the algorithm are the calls to the randomized cycle cover approximation algorithm in line 5. Using amplification we can assume that the probability that all the calls to this algorithm succeed is at least $1/2$.

It remains to show that if the algorithm Alg-k-MaxTSP succeeds, it outputs some $(1-1/c)$-approximate set of Hamiltonian cycles. Hence, for the remainder of the proof, let us assume that the algorithm and hence all calls to the internal FPRAS succeed. Furthermore, let $R \subseteq E$ be some Hamiltonian cycle of G. We will argue that there is some iteration where the algorithm outputs an $(1 - 1/c)$-approximation of R.

For each $1 \leq i \leq k$, let $F_{H,i} \subseteq R$ be some set of $3\,c\,k$ heaviest edges of R in the i-th component, breaking ties arbitrarily. Let $F_H = \bigcup_{i=1}^{k} F_{H,i}$. We define $F_L \subseteq R$ such that $F_L \cap F_H = \emptyset$ and each edge in F_H is part of a path in $F_L \cup F_H$ that contains c edges from F_L. This is always possible as long as R is large enough. We now have $\#F_H \leq 3\,c\,k^2$ and $\#F_L \leq c\#F_H$. Hence in line 1 there will be some iteration that chooses F_H and F_L. We fix

this iteration for the remainder of the proof.

Let $\delta \in \mathbb{N}^k$ as defined in line 2 and observe that $\delta_i = \min\{w_i(e) \mid e \in F_{H,i}\}$ for all i, which means that for all edges $e \in R \setminus F_H$ we have $w(e) \leq \delta$. Hence the loop in line 3 sets the weights of all edges $e \in E \setminus R$ that do not fulfill $w(e) \leq \delta$ to zero, and these are the only weights that are modified. In particular, this does not affect edges in R, hence $w(R)$ remains unchanged. Note that since we do not increase the weight of any edge and do not change the weight of the edges in R, it suffices to show that the algorithm computes an approximation with respect to the changed weights.

Next we obtain a $(1-1/\#V)$-approximate set $\mathcal{S}$ of c-cycle covers of G that contain $F_H \cup F_L$. Since R is a c-cycle cover of G with $F_H \cup F_L \subseteq R$, there must be some c-cycle cover $S \in \mathcal{S}$ with $F_H \cup F_L \subseteq S$ that $(1 - 1/\#V)$-approximates R. Hence in line 6 there will be some iteration that chooses this S. Again we fix this iteration for the remainder of the proof.

As in line 7, let $C_1, \ldots, C_r$ denote the cycles in S. Note that each cycle contains at least c edges. Since each edge in F_H is part of a path in $F_H \cup F_L$ with at least c edges from F_L, we even know that each cycle contains at least c edges not from F_H and thus the condition in line 8 is fulfilled. Let these edges $e_{i,j}$ be defined as in the algorithm. Note that since $e_{i,j} \notin F_H$ we have $w(e_{i,j}) \leq \delta$ for all i, j, because the weight function was changed accordingly.

In line 10 we compute some $\chi\colon \{1, \ldots, r\} \to \{1, \ldots, c\}$ such that

$$\sum_{i=1}^{r} w(e_{i,\chi(i)}) \quad \leq \quad 2\,k \cdot \delta + \frac{1}{c} \sum_{i=1}^{r} \sum_{j=1}^{c} w(e_{i,j}) \leq 2\,k \cdot \delta + \frac{1}{c} \cdot w(S \setminus F_H).$$

Recall that by Corollary 3 such a coloring exists and can be computed in polynomial time. Removing the chosen edges breaks the cycles into simple paths, which can be arbitrarily connected to a Hamiltonian cycle R'. For the following estimation note that $\delta \leq \frac{w(F_H)}{3\,c\,k}$ and $w(F_H) \geq \frac{3\,c\,k}{m} w(R)$ and recall that $m = \#V = \#R$.

$$\begin{aligned} w(R') \quad &\geq \quad w(S) - \sum_{i=1}^{r} w(e_{i,\chi(i)}) \geq w(S) - 2\,k \cdot \delta - \frac{1}{c} \cdot w(S \setminus F_H) \\ &= \quad \left(1 - \frac{1}{c}\right) w(S) + \frac{1}{c} w(F_H) - 2\,k \cdot \delta \geq \left(1 - \frac{1}{c}\right) w(S) + \frac{1}{3\,c} w(F_H) \\ &\geq \quad \left(1 - \frac{1}{c}\right) \left(1 - \frac{1}{m}\right) w(R) + \frac{k}{m} w(R) \geq \left(1 - \frac{1}{c}\right) w(R) \end{aligned}$$

This proves the assertion. ◀

It is known that 1-c-MaxDCC is APX-hard for all $c \geq 3$ [4] and that 1-c-MaxUCC is APX-hard for $c \geq 5$ [11]. This means that, unless P = NP, there is no PTAS for these problems (and especially not for the variants with fixed edges). Furthermore, the existence of an FPRAS or PRAS for these problems implies NP = RP and thus a collapse of the polynomial-time hierarchy, which is seen as follows.

If an APX-hard problem has a PRAS, then all problems in APX have a PRAS and hence MAX-3SAT has one. There exists an $\varepsilon > 0$ and a polynomial-time computable f mapping CNF formulas to 3-CNF formulas such that if $x \in$ SAT, then $f(x) \in$ 3SAT; and if $x \notin$ SAT, then there is no assignment satisfying more than a fraction of $1 - \varepsilon$ of $f(x)$'s clauses [1, Theorem 10.1]. The PRAS for MAX-3SAT allows us to compute probabilistically a $(1 - \varepsilon/2)$-approximation for $f(x)$ which in turn tells us whether or not $x \in$ SAT. Since this procedure has no false negatives we get RP = NP, which implies a collapse of the polynomial-time hierarchy [10, 17].

So it seems unlikely that there is a PRAS for 1-c-MaxDCC where $c \geq 3$ and 1-c-MaxUCC where $c \geq 5$. However, this does not necessarily mean that the above algorithm is useless for parameters $c \geq 3$ in the directed and $c \geq 5$ in the undirected case: The algorithm could still benefit from a constant-factor approximation for k-c-MaxUCC$_{\mathrm{F}}$ or k-c-MaxDCC$_{\mathrm{F}}$. A simple change in the estimation shows that if the cycle cover algorithm has an approximation ratio of α, the above algorithm provides an approximation with ratio $\alpha(1 - 1/c)$.

▶ **Theorem 5.** *Let $k \geq 1$.*

1. *k-MaxATSP is randomized $1/2$-approximable.*
2. *k-MaxSTSP is randomized $2/3$-approximable.*

Proof. We combine Theorem 1 and Lemma 4. ◀

5 Approximating Multiobjective Maximum Satisfiability

5.1 Definition

Given a formula in CNF and a function that maps each clause to a k-objective weight, our goal is to find truth assignments that maximize the sum of the weights of all satisfied clauses. The formal definition is as follows.

k-Objective Maximum Weighted Satisfiability (k-MaxSAT)
Instance: Formula H in CNF over a set of variables V, weight function $w\colon H \to \mathbb{N}^k$
Solution: Truth assignment $I\colon V \to \{0,1\}$
Weight: Sum of the weights of all clauses satisfied by I, i.e., $w(I) = \sum_{\substack{C \in H \\ I(C)=1}} w(C)$

5.2 Previous Work

The first approximation algorithm for maximum satisfiability is due to Johnson [8], whose greedy algorithm showed that the single-objective 1-MaxSAT problem is $1/2$-approximable. Further improvements on the approximation ratio followed, and the currently best known approximation ratio of 0.7846 for 1-MaxSAT is due to Asano and Williamson [2].

Only little is known about k-MaxSAT for $k \geq 2$. Santana et. al. [16] apply genetic algorithms to a version of the problem that is equivalent to k-MaxSAT with polynomially bounded weights. To our knowledge, the approximability of k-MaxSAT for $k \geq 2$ has not been investigated so far.

5.3 Our Results

We show that k-MaxSAT is $1/2$-approximable mainly by transferring the idea that for any truth assignment, the assignment itself or its complementary assignment satisfies at least one half of all clauses to multidimensional objective functions. We choose some suitable parameter $l \in \mathbb{N}$ depending only on the number of objectives. For a given formula in CNF we try all possible partial truth assignments for each set of at most l variables using brute force and extend each partial assignment to a full assignment in the following way: For each remaining variable v we compute two vectors roughly representing the weight gained by the two possible assignments for v. We then compute a 2-coloring of those weight vectors with low discrepancy which completes the partial assignment to a truth assignment whose weight is at least one half of the total weight of the remaining satisfiable clauses minus some error.

This error can be compensated by choosing l large enough such that the partial assignment already contributes a large enough weight. This results in a $1/2$-approximation for k-MaxSAT.

For a set of clauses H and a variable v let $H[v=1]=\{C\in H \mid v\in C\}$ be the set of clauses that are satisfied if this variable is assigned one, and analogously $H[v=0]=\{C\in H \mid \overline{v}\in C\}$ be the set of clauses that are satisfied if this variable is assigned zero. This notation is extended to sets of variables V by $H[V=i]=\bigcup_{v\in V} H[v=i]$ for $i=0,1$.

Algorithm: Alg-k-MaxSAT(H,w)

Input : Formula H in CNF over the variables $V=\{v_1,\dots,v_m\}$, k-objective weight function $w\colon H\to\mathbb{N}^k$

Output : Set of truth assignments $I\colon V\to\{0,1\}$

1 `foreach` disjoint $V^0,V^1\subseteq V$ with $\#(V^0\cup V^1)\leq 4k^2$ `do`

2 $\quad G:=H\setminus(H[V^0=0]\cup H[V^1=1])$;

3 $\quad \hat{V}^{(1-i)}:=\{v\in V\setminus(V^0\cup V^1)\mid 4k\cdot w(G[v=i])\not\leq w(H\setminus G)\}$, $i=0,1$;

4 $\quad$ `if` $\hat{V}^0\cap\hat{V}^1=\emptyset$ `then`

5 $\quad\quad V':=V\setminus(V^0\cup V^1\cup\hat{V}^0\cup\hat{V}^1)$, $L':=V'\cup\{\overline{v}\mid v\in V'\}$;

6 $\quad\quad G':=(G[V'=0]\cup G[V'=1])\setminus(G[\hat{V}^0=0]\cup G[\hat{V}^1=1])$;

7 $\quad\quad$ for $v_j\in V'$ let $x^{j,i}=\sum\{\frac{w(C)}{\#(C\cap L')}\mid C\in G'[v_j=i]\}$ for $i=0,1$;

8 $\quad\quad$ compute some coloring $\chi\colon V'\to\{0,1\}$ such that

$$\sum_{v_j\in V'} x^{j,\chi(j)}\geq\frac{1}{2}\sum_{v_j\in V'}(x^{j,0}+x^{j,1})-2k\delta$$

$\quad\quad$ where $\delta_r=\max\{x_r^{j,i}\mid v_j\in V', i\in\{0,1\}\}$;

9 $\quad\quad$ let $I(v):=i$ for $v\in V^i\cup\hat{V}^i\cup\chi^{-1}(\{i\})$, $i=0,1$;

10 $\quad\quad$ output I

▶ **Theorem 6.** *k-MaxSAT is $1/2$-approximable for any $k\geq 1$.*

Proof. We show that this approximation is realized by Alg-k-MaxSAT. First note that this algorithm runs in polynomial time since k is constant and the coloring in line 8 can be computed in polynomial time using Corollary 3. For the correctness, let (H,w) be the input where H is a formula over the variables $V=\{v_1,\dots,v_m\}$ and $w\colon H\to\mathbb{N}^k$ is the k-objective weight function. Let $I_o\colon V\to\{0,1\}$ be an optimal truth assignment. We show that there is a loop iteration of Alg-k-MaxSAT(H,w) that outputs a truth assignment I such that $w(I)\geq w(I_o)/2$.

We first note that there are sets V^0 and V^1 with a bounded cardinality of at most $4k^2$ that define a partial truth assignment that contributes a large weight.

▶ **Claim 7.** *There are sets $V^i\subseteq I_o^{-1}(\{i\})$, $i=0,1$ with $\#(V^0\cup V^1)\leq 4k^2$ such that for $G=H\setminus(H[V^0=0]\cup H[V^1=1])$ and any $v\in V\setminus(V^0\cup V^1)$ it holds that*

$$w(G[v=I_o(v)])\leq\frac{1}{4k}w(H\setminus G). \tag{1}$$

Proof Sketch. The set $V^0\cup V^1$ is obtained by iteratively choosing variables such that (one component of) the weight of the remaining clauses that get satisfied if the variable is set to its value under I_o is high, while the components are chosen in a round-robin fashion. Since $V^0\cup V^1$ contains the $4k$ most "influential" variables per objective, none of the remaining variables can have high "influence" on the remaining clauses, because otherwise one of them would have been chosen to belong to $V^0\cup V^1$. ◀

We choose the iteration of the algorithm where V^0 and V^1 equal the sets whose existence is guaranteed by Claim 7. In the following, we use the variables as they are defined in the algorithm. Observe that by the claim it holds that $I_o(v) = i$ for all $v \in \hat{V}^i$ for $i = 0, 1$ and thus $\hat{V}^0 \cap \hat{V}^1 = \emptyset$. Note that

$$\begin{aligned}\sum_{v_j \in V'} x^{j,0} + x^{j,1} &= \sum_{v_j \in V'} \sum_{i \in \{0,1\}} \sum_{C \in G'[v_j = i]} \frac{w(C)}{\#(C \cap L')} \\ &= \sum_{C \in G'} \#(C \cap L') \frac{w(C)}{\#(C \cap L')} \\ &= w(G').\end{aligned}$$

Furthermore, for all $v_j \in V'$ and $i = 0, 1$, we have the bound $x^{j,i} \leq w(G'[v_j = i]) \leq w(G[v_j = i]) \leq \frac{1}{4k} w(H \setminus G)$ because of the definition of V' and $\hat{V}^{(1-i)}$. By Corollary 3, we find a coloring $\chi \colon V' \to \{0, 1\}$ such that for each $1 \leq i \leq k$ it holds that

$$\left| \frac{1}{2} \sum_{v_j \in V'} \sum_{r=0}^{1} x_i^{j,r} - \sum_{v_j \in V'} x_i^{j,\chi(v_j)} \right| \leq 2k \max_{j,r} |x_i^{j,r}| \leq 2k \frac{1}{4k} w_i(H \setminus G) = \frac{1}{2} w_i(H \setminus G)$$

and hence

$$\sum_{v_j \in V'} x^{j,\chi(v_j)} \geq \frac{1}{2} \sum_{v_j \in V'} (x^{j,0} + x^{j,1}) - \frac{1}{2} w(H \setminus G) = \frac{1}{2}(w(G') - w(H \setminus G)).$$

For I being the truth assignment generated in this iteration it holds that

$$w(\{C \in G' \mid I(C) = 1\}) \geq \sum_{v_j \in V'} x^{j,\chi(v_j)} \geq \frac{1}{2}(w(G') - w(H \setminus G)). \tag{2}$$

Furthermore, since I and I_o coincide on $V \setminus V'$, we have

$$\begin{aligned} w(\{C \in H \setminus G' \mid I(C) = 1\}) &= w(\{C \in H \setminus G' \mid I_o(C) = 1\}) && (3) \\ &\geq w(\{C \in H \setminus G \mid I_o(C) = 1\}) \\ &= w(\{H \setminus G\}). && (4)\end{aligned}$$

Thus we finally obtain

$$\begin{aligned} w(I) &= w(\{C \in H \setminus G' \mid I(C) = 1\}) + w(\{C \in G' \mid I(C) = 1\}) \\ &\overset{(2)}{\geq} w(\{C \in H \setminus G' \mid I(C) = 1\}) + \tfrac{1}{2}(w(G') - w(H \setminus G)) \\ &\overset{(3)}{=} w(\{C \in H \setminus G' \mid I_o(C) = 1\}) + \tfrac{1}{2}(w(G') - w(H \setminus G)) \\ &\overset{(4)}{\geq} \tfrac{1}{2} w(\{C \in H \setminus G' \mid I_o(C) = 1\}) + \tfrac{1}{2} w(G') \\ &\geq \tfrac{1}{2} w(I_o). \end{aligned}$$

◀

References

1 S. Arora and C. Lund. Hardness of approximations. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, Boston, 1997.

2 T. Asano and D. P. Williamson. Improved approximation algorithms for MAX SAT. *Journal of Algorithms*, 42(1):173–202, 2002.

3 J. Beck and T. Fiala. "Integer-Making" Theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981.

4 M. Bläser and B. Manthey. Approximating maximum weight cycle covers in directed graphs with weights zero and one. *Algorithmica*, 42(2):121–139, 2005.

5 B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability & Computing*, 12(4):365–399, 2003.

6 M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for finding a maximum weight Hamiltonian circuit. *Operations Research*, 27(4):799–809, 1979.

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

8 D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer System Sciences*, 9(3):256–278, 1974.

9 H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *Journal of the ACM*, 52(4):602–626, 2005.

10 C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17:215–217, 1983.

11 B. Manthey. On approximating restricted cycle covers. *SIAM J. Comput.*, 38(1):181–206, 2008.

12 B. Manthey. On approximating multi-criteria TSP. In S. Albers and J.-Y. Marion, editors, *26th Intern. Symposium on Theoretical Aspects of Computer Science, STACS 2009*, pages 637–648. Dagstuhl Research Online Publication Server, 2009.

13 B. Manthey. Deterministic algorithms for multi-criteria TSP. In *Proceedings of the International Conference on Theory and Applications of Models of Computation*, volume 6648 of *Lecture Notes in Computer Science*, pages 264–275. Springer Verlag, 2011.

14 B. Manthey and L. S. Ram. Approximation algorithms for multi-criteria traveling salesman problems. *Algorithmica*, 53(1):69–88, 2009.

15 K. Paluch, M. Mucha, and A. Madry. A 7/9 - approximation algorithm for the maximum traveling salesman problem. In I. Dinur, K. Jansen, J. Naor, and J. Rolim, editors, *Proceedings of APPROX/RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 298–311. Springer Berlin / Heidelberg, 2009.

16 R. Santana, C. Bielza, J. A. Lozano, and P. Larrañaga. Mining probabilistic models learned by EDAs in the optimization of multi-objective problems. In *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 445–452, New York, NY, USA, 2009. ACM.

17 M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th Symposium on Theory of Computing*, pages 330–335, 1983.

Quasi-Weak Cost Automata: A New Variant of Weakness *

Denis Kuperberg[1] and Michael Vanden Boom[2]

1 **LIAFA/CNRS/Université Paris 7, Denis Diderot, France**
`denis.kuperberg@liafa.jussieu.fr`
2 **Department of Computer Science, University of Oxford**
Wolfson Building, Parks Road, Oxford OX1 3QD, England
`michael.vandenboom@cs.ox.ac.uk`

Abstract

Cost automata have a finite set of counters which can be manipulated on each transition but do not affect control flow. Based on the evolution of the counter values, these automata define functions from a domain like words or trees to $\mathbb{N} \cup \{\infty\}$, modulo an equivalence relation which ignores exact values but preserves boundedness properties. These automata have been studied by Colcombet et al. as part of a "theory of regular cost functions", an extension of the theory of regular languages which retains robust equivalences, closure properties, and decidability like the classical theory.

We extend this theory by introducing quasi-weak cost automata. Unlike traditional weak automata which have a hard-coded bound on the number of alternations between accepting and rejecting states, quasi-weak automata bound the alternations using the counter values (which can vary across runs). We show that these automata are strictly more expressive than weak cost automata over infinite trees. The main result is a Rabin-style characterization theorem: a function is quasi-weak definable if and only if it is definable using two dual forms of non-deterministic Büchi cost automata. This yields a new decidability result for cost functions over infinite trees.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Automata, infinite trees, cost functions, weak

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.66

1 Introduction

Cost automata are finite-state machines enriched with counters which can be manipulated on each transition but cannot be used to affect control flow. Based on the evolution of the counter values, these automata define functions from some domain (like words or trees over a finite alphabet) to $\mathbb{N} \cup \{\infty\}$, modulo an equivalence relation $\approx$ which ignores exact values but preserves boundedness properties. By only considering the functions up to $\approx$, the resulting "theory of regular cost functions" retains many of the equivalences, closure properties, and decidability results of the theory of regular languages [3]. It extends the classical theory since we can identify each language with its characteristic function mapping structures in the language to 0 and everything else to ∞; it is a strict extension since cost automata can count some behaviour within the input structure.

* The research leading to these results has received funding from ANR 2010 BLAN 0202 02 FREC and the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement 259454.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 66–77

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The development of this theory was motivated by problems which can be reduced to questions of boundedness. For instance, Hashiguchi [7] and later Kirsten [8] used distance and nested-distance desert automata (special forms of cost automata) to prove the decidability of the "star-height problem": given a regular language L of finite words and a natural number n, is there some regular expression (using concatenation, union, and Kleene star) for L which uses at most n nestings of Kleene-star operations? Colcombet and Löding [4] have used a similar approach over finite trees. The theory of regular cost functions over finite words [3] and finite trees [6] can be viewed as a unifying framework for these problems.

It is desirable to extend the theory to infinite trees. For instance, the "parity-index problem" asks: given a regular language L of infinite trees and $i < j$, is there a parity automaton using only priorities $\{i, i+1, \ldots, j\}$? This is known to be decidable in some special cases (e.g. for deterministic languages [11]), but a general decision procedure is not known. However, Colcombet and Löding [5] have reduced the parity-index problem to another open problem, namely the decidability of $\approx$ for regular cost functions over infinite trees.

Weak cost automata (recently studied in [13]) are a natural starting point in this line of research on regular cost functions over infinite trees. In the classical setting, weak automata are a restricted form of alternating Büchi automata which have a fixed bound on the number of alternations between accepting and rejecting states across all runs. They were introduced in [10] to characterize the languages definable in weak monadic second-order logic (WMSO), a variant of MSO in which second-order quantifiers are interpreted over finite sets. Prior to this work, Rabin [12] had given an interesting characterization using non-deterministic automata, showing that a language is weakly definable if and only if the language and its complement are non-deterministic Büchi recognizable.

These notions of weakness have received considerable attention because the weakly definable languages are expressive (e.g. they capture CTL), but still admit efficient model-checking [9]. Indeed, in order to improve the efficiency in some model-checking scenarios, Kupferman and Vardi [9] adapted Rabin's work and provided a quadratic translation between non-deterministic Büchi automata and the corresponding weak automaton.

In [13], weak cost automata were shown equivalent to "cost WMSO", in analogy to the classical theory. However, the question of a Rabin-style characterization based on non-deterministic automata remained open and prompted this work.

1.1 Contributions

We continue the study of regular cost functions over infinite trees by introducing a variant of weakness which we call "quasi-weakness". Unlike traditional weak automata which have a hard-coded bound on the number of alternations between accepting and rejecting states, quasi-weak automata bound the alternations using counter values (which can vary across runs). We show that quasi-weak cost automata are strictly more expressive than weak cost automata over infinite trees.

Although there is no notion of complement for a function, there are two dual semantics (B and S) used to define cost functions. We show that quasi-weak B-automata can be simulated by non-deterministic B-Büchi and S-Büchi automata. Combined with results from [13], this implies the decidability of $f \approx g$ when f, g are cost functions defined by quasi-weak B-automata. Consequently, this work extends the class of cost functions over infinite trees for which $\approx$ is known to be decidable. We also provide a non-trivial extension of Kupferman and Vardi's construction [9] to translate equivalent non-deterministic B-Büchi and S-Büchi automata to an equivalent quasi-weak B-automaton (where the equivalence in each case is up to $\approx$). This provides a Rabin-style characterization of the functions definable

using quasi-weak B-automata and marks an interesting departure from the classical theory.

The construction relies on analyzing a composed run of a B-Büchi automaton and S-Büchi automaton. To aid in this analysis, we use BS-automata and introduce a corresponding BS-equivalence relation $\cong$ which can be used to compare cost automata which define not one but two functions (one based on the B-counters and one on the S-counters). Although the B- and S-counter actions in such a composed run can be independent, we show that it is possible to effectively construct an equivalent (up to $\cong$) BS-automaton in which the actions are more structured (namely, the counters are "hierarchical" so they can be totally ordered and manipulating a higher counter resets all lower counters). We believe this may be a useful technique in other situations which require both counter types.

1.2 Organization

We define cost automata on infinite trees in Sect. 2, with semantics based on two-player infinite games. We also introduce the new quasi-weak cost automata and compare to the more traditional weak cost automata. In Sect. 3, we consider automata with both counter types and show how they can be made hierarchical. Finally, in Sect. 4, we describe the other components of the main result, a Rabin-style characterization for quasi-weak cost automata. We conclude with some open questions in Sect. 5.

1.3 Notations

We write $\mathbb{N}$ for the set of non-negative integers and $\mathbb{N}_\infty$ for the set $\mathbb{N} \cup \{\infty\}$, ordered by $0 < 1 < \cdots < \infty$. If $i \leq j$ are integers, $[i, j]$ denotes the set $\{i, i+1, \ldots, j\}$. We fix a finite alphabet $\mathbb{A}$. The set of finite (respectively, infinite) words over $\mathbb{A}$ is $\mathbb{A}^*$ (respectively, $\mathbb{A}^\omega$) and the empty word is ϵ. For notational simplicity we work only with infinite binary trees. Let $\mathcal{T} = \{0,1\}^*$ be the unlabelled infinite binary tree. A *branch* in $\mathcal{T}$ is a word $\pi \in \{0,1\}^\omega$. The set $\mathcal{T}_\mathbb{A}$ of *complete $\mathbb{A}$-labelled binary trees* is composed of mappings $t : \mathcal{T} \to \mathbb{A}$.

Non-decreasing functions $\mathbb{N} \to \mathbb{N}$ will be denoted by letters $\alpha, \beta, \ldots$, and will be extended to $\mathbb{N}_\infty$ by $\alpha(\infty) = \infty$. We call these *correction functions*.

2 Cost Automata

2.1 Cost Functions

Let E be any set, and $\mathcal{F}_E$ be the set of functions $: E \to \mathbb{N}_\infty$. For $f, g \in \mathcal{F}_E$ and α a correction function, we write $f \preccurlyeq_\alpha g$ if $f \leq \alpha \circ g$ (or if we are comparing single values $n, m \in \mathbb{N}$, $n \preccurlyeq_\alpha m$ if $n \leq \alpha(m)$). We write $f \approx_\alpha g$ if $f \preccurlyeq_\alpha g$ and $g \preccurlyeq_\alpha f$. Finally, $f \approx g$ (respectively, $f \preccurlyeq g$) if $f \approx_\alpha g$ (respectively, $f \preccurlyeq_\alpha g$) for some α. The idea is that the *boundedness relation* $\approx$ does not pay attention to exact values, but does preserve the existence of bounds. Remark that $f \not\preccurlyeq g$ if and only if there exists a set $D \subseteq E$ such that g is bounded on D but f is unbounded on D.

A *cost function over E* is an equivalence class of $\mathcal{F}_E/\approx$. In practice, a cost function (denoted $f, g, \ldots$) will be represented by one of its elements in $\mathcal{F}_E$. In this paper, E will usually be $\mathcal{T}_\mathbb{A}$. The functions defined by automata will always be considered as cost functions, i.e. only considered up to $\approx$.

2.2 B- and S-Valuations

Cost automata define functions from $\mathcal{T}_{\mathbb{A}}$ to $\mathbb{N}_\infty$. The valuation is based on both classical acceptance conditions (in this paper, Büchi acceptance) and a finite set of counters Γ.

A counter γ is initially assigned value 0 and can be *incremented* $\mathtt{i}$, *reset* $\mathtt{r}$ to 0, *checked* $\mathtt{c}$, or left unchanged ε. Given an infinite word u_γ over the alphabet $\{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{c}\}$, we define a set $C(u_\gamma) \subseteq \mathbb{N}$ which collects the checked values of γ. In the case of a finite set of counters Γ and a word u over $\{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{c}\}^\Gamma$, $C(u) := \bigcup_{\gamma\in\Gamma} C(pr_\gamma(u))$ ($pr_\gamma(u)$ is the γ-projection of u).

We will separate counters into two types: B-counters, which accept as atomic actions the set $\mathbb{B} = \{\varepsilon, \mathtt{ic}, \mathtt{r}\}$, and S-counters, with atomic actions $\mathbb{S} = \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$. Given B-counters Γ_B and $u \in (\mathbb{B}^{\Gamma_B})^\omega$, the *$B$-valuation* is $val_B(u) := \sup C(u)$; likewise, given S-counters Γ_S and $u \in (\mathbb{S}^{\Gamma_S})^\omega$, the *$S$-valuation* is $val_S(u) := \inf C(u)$. By convention, $\inf \emptyset = \infty$ and $\sup \emptyset = 0$. For instance $val_B((\mathtt{ic})^\omega) = \infty$, $val_B((\mathtt{icr})^\omega) = 1$, $val_S(\mathtt{i}^{100}\mathtt{cri}\varepsilon\mathtt{icr}(\mathtt{r})^\omega) = 2$, and $val_S(\mathtt{i}^\omega) = \infty$ because the counter is never checked.

In all cases, if the set of counters Γ is $[1, k]$, an action ν is called *hierarchical* if there is some $i \in [1, k]$ such the action ν performs ε on all counters $j > i$, and $\mathtt{r}$ on all counters $j < i$. It means that performing an increment or a reset on counter i resets all counters j below it.

Cost automata are named B-, S-, or BS-automata depending on the type(s) of counters used. They are *hierarchical* (written, e.g. hB-automata) if only hierarchical actions are used.

2.3 B- and S-Automata on Infinite Trees

An *alternating B-Büchi automaton* $\mathcal{A} = \langle Q, \mathbb{A}, q_0, F, \Gamma_B, \delta\rangle$ on infinite trees has a finite set of states Q, alphabet $\mathbb{A}$, initial state $q_0 \in Q$, accepting states F, finite set Γ_B of B-counters, and transition function $\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\{0, 1\} \times \mathbb{B}^{\Gamma_B} \times Q)$, where $\mathcal{B}^+(\{0, 1\} \times \mathbb{B}^{\Gamma_B} \times Q)$ is the set of positive boolean combinations, written as a disjunction of conjunctions of elements $(d, \nu, q) \in \{0, 1\} \times \mathbb{B}^{\Gamma_B} \times Q$. *Alternating S-Büchi automata* are defined in the same way, replacing B-counters by S-counters and $\mathbb{B}$ with $\mathbb{S}$.

We view running a B-automaton (resp. S-automaton) $\mathcal{A}$ on an input tree t as a game $(\mathcal{A}, t)$ between two players : Eve is in charge of the disjunctive choices and tries to minimize (resp. maximize) counter values while satisfying the Büchi condition, and Adam is in charge of the conjunctive choices and tries to maximize (resp. minimize) counter values or show the Büchi condition is not satisfied. Because the transition function is given as a disjunction of conjunctions, we can consider that at each position, Eve first chooses a disjunct, and then Adam chooses a single tuple (d, ν, q) in this disjunct.

A *play* of $\mathcal{A}$ on input t is a sequence $q_0, (d_1, \nu_1, q_1), (d_2, \nu_2, q_2), \dots$ compatible with t and δ, i.e. q_0 is initial, and for all $i \in \mathbb{N}$, $(d_{i+1}, \nu_{i+1}, q_{i+1})$ appears in $\delta(q_i, t(d_1 \dots d_i))$.

A *strategy* for Eve (resp. Adam) in the game $(\mathcal{A}, t)$ is a function that fixes the next choice of Eve (resp. Adam), based on the history of the play (resp. the history of the play and Eve's choice of disjunct). A strategy is *finite-memory* if the number of memory states needed for the player to choose the next move is finite. A strategy is *positional* if no memory at all is needed: the player only needs to know the current position. Notice that choosing a strategy for Eve and a strategy for Adam fixes a play in $(\mathcal{A}, t)$. We say a play π is *compatible* with a strategy σ for Eve if there is some strategy σ' for Adam such that σ and σ' yield the play π.

A play π is *accepting* if there is $q \in F$ appearing infinitely often in π (i.e. π satisfies the Büchi acceptance condition). Given a play π from a B-automaton $\mathcal{A}$, the value of π is $val(\pi) := val_B(h_B(\pi))$ if π is accepting, and $val(\pi) = \infty$ otherwise (where h_B is the projection of π to the B-actions). This yields the maximum checked counter value if

the play is accepting, and ∞ otherwise. We assign a value to a strategy σ for Eve by $val(\sigma) := \sup\{val(\pi) : \pi \text{ is compatible with } \sigma\}$. The value of $\mathcal{A}$ over a tree t is $[\![\mathcal{A}]\!]_B(t) := \inf\{val(\sigma) : \sigma \text{ is a strategy for Eve in the game } (\mathcal{A}, t)\}$.

Likewise, in an S-automaton $\mathcal{A}'$, we define $val(\pi) := val_S(h_S(\pi))$ if π is accepting, and 0 otherwise (where h_S is the projection to the S-actions). Once again, counter actions are only considered if the play is accepting (this time the minimum checked value is used), and 0 is assigned to rejecting plays. Then $val(\sigma) := \inf\{val(\pi) : \pi \text{ is compatible with } \sigma\}$, and $[\![\mathcal{A}']\!]_S(t) := \sup\{val(\sigma) : \sigma \text{ is a strategy for Eve in the game } (\mathcal{A}', t)\}$.

We consider $[\![\mathcal{A}]\!]_B$ and $[\![\mathcal{A}']\!]_S$ as cost functions, so we always work modulo the cost function equivalence $\approx$. If it is clear what semantic the automaton uses we will omit the subscript and write just $[\![\mathcal{A}]\!]$ or $[\![\mathcal{A}']\!]$. If $f \approx [\![\mathcal{A}]\!]$ then we say $\mathcal{A}$ *recognizes* the cost function f.

If for all $(q, a) \in Q \times \mathbb{A}$, $\delta(q, a)$ is of the form $\bigvee_i (0, \nu_i, q_i) \wedge (1, \nu'_i, q'_i)$, then we say the automaton is *non-deterministic*. We define a *run* to be the set of possible plays compatible with some fixed strategy of Eve. Since the only choices of Adam are in the branching, a run labels the entire binary tree with states, and choosing a branch yields a unique play of the automaton. A run is accepting if all of its plays are accepting (that is, if it is accepting on all branches). A value is assigned to a run of a B-automaton (resp. S-automaton) by taking the supremum (resp. infimum) of the values across all branches.

Finally, a cost automaton $\mathcal{A} = \langle Q, \mathbb{A}, q_0, F, \Gamma, \delta \rangle$ is *weak* if the state-set Q can be partitioned into $Q_1, \ldots, Q_k$ satisfying:

- for all i and for all $q, q' \in Q_i$, $q \in F$ if and only if $q' \in F$;
- if some (d, ν, q) appears in some $\delta(p, a)$ with $p \in Q_i$ and $q \in Q_j$, then $j \leq i$.

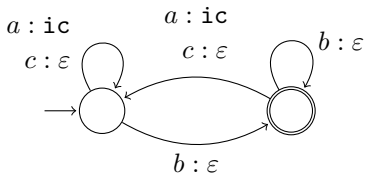
This means there is a fixed bound k on the number of alternations between accepting and rejecting states, so any accepting play must stabilize in an accepting partition.

2.3.1 Examples

Let $\mathbb{A} = \{a, b, c\}$ and let f be the cost function over $\mathbb{A}$-labelled trees where $f(t) = \infty$ if there is a branch with only finitely many b's, and $f(t) = \sup\{|\pi|_a : \pi \text{ is a branch of } t\}$ otherwise, where $|\pi|_a$ denotes the number of a's in π.

We define a non-deterministic B-Büchi automaton $\mathcal{U}$ and a non-deterministic S-Büchi automaton $\mathcal{U}'$, together with a weak automaton $\mathcal{B}$, such that $f \approx [\![\mathcal{U}]\!] \approx [\![\mathcal{U}']\!] \approx [\![\mathcal{B}]\!]$.

The principle of $\mathcal{U}$ is to simultaneously count a's and check for infinitely many b's by running the following deterministic B-automaton on every branch. We write $a : \nu$ to denote that on input a, the counter action is ν; accepting states are denoted by double circles.

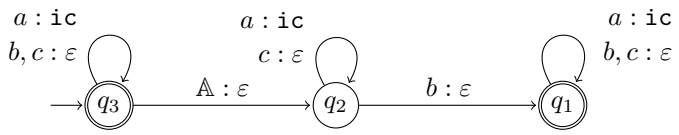


On the other hand, $\mathcal{U}' = \langle \{p_a, p_b, q_b, \top\}, \mathbb{A}, \{p_a, p_b\}, \{q_b, \top\}, \{\gamma\}, \delta \rangle$ tries to find a branch π with either a lot of a's (state p_a), or only finitely many b's (state p_b), in order to witness a high value for f (∞ in the second case). For simplicity, we allow here two initial states, but this does not add expressive power to the model. The state q_b is used when Eve has guessed the position of the last b, and still needs to prove that there are no more b on π, and $\top$ is used when the remainder of the branch does not matter.

The transition table δ for $\mathcal{U}'$ follows. Remark that $\mathcal{U}'$ is in fact a non-deterministic weak S-automaton.

δ	p_a	p_b	q_b	$\top$
a	$((0,\mathtt{i},p_a)\wedge(1,\varepsilon,\top))$ $\vee((0,\varepsilon,\top)\wedge(1,\mathtt{i},p_a))$ $\vee((0,\mathtt{cr},\top)\wedge(1,\varepsilon,\top))$ $\vee((0,\varepsilon,\top)\wedge(1,\mathtt{cr},\top))$	$((0,p_b)\wedge(1,\varepsilon,\top))$ $\vee((0,\varepsilon,\top)\wedge(1,\varepsilon,p_b))$ $\vee((0,\varepsilon,q_b)\wedge(1,\varepsilon,\top))$ $\vee((0,\varepsilon,\top)\wedge(1,\varepsilon,q_b))$	$((0,\varepsilon,q_b)\wedge(1,\varepsilon,\top))$ $\vee((0,\varepsilon,\top)\wedge(1,\varepsilon,q_b))$	$(0,\varepsilon,\top)\wedge(1,\varepsilon,\top)$
b	$((0,\varepsilon,p_a)\wedge(1,\varepsilon,\top))$ $\vee((0,\varepsilon,\top)\wedge(1,\varepsilon,p_a))$ $\vee((0,\mathtt{cr},\top)\wedge(1,\varepsilon,\top))$ $\vee((0,\varepsilon,\top)\wedge(1,\mathtt{cr},\top))$	$=\delta(p_b,a)$	*false* (empty disjunction)	$(0,\varepsilon,\top)\wedge(1,\varepsilon,\top)$
c	$=\delta(p_a,b)$	$=\delta(p_b,a)$	$=\delta(q_b,a)$	$(0,\varepsilon,\top)\wedge(1,\varepsilon,\top)$

Finally, $\mathcal{B}$ is designed such that Adam controls all of the choices: Adam selects a single branch, and runs the following automaton on this branch (he controls the non-determinism):



If there is a branch π with finitely many b's, Adam can select π and stabilize in rejecting state q_2 by moving from q_1 to q_2 after the last b. This witnesses value ∞ for f. Otherwise, Adam tries to select a branch which maximizes the number of a's. The state-set can be partitioned such that $Q_i = \{q_i\}$ for $i \in [1,3]$.

2.4 Quasi-Weak B-Automata

We want to define an extension of weak B-automata, which preserves the property that accepting plays must stabilize in accepting states. The idea of weak automata is to bound the number of alternations between accepting and rejecting states by a hard bound.

Here we have another available tool to bound the number of such alternations: the counters. We know that in a B-automaton, an accepting play of finite value n does at most n increments between resets, but this number is not known a priori by the automaton. Thus, if we guarantee there is correction function α such that in any play π of value n, $\alpha(n)$ is greater than the number of alternations between accepting and rejecting states in π, then we know that any play of finite value must stabilize in accepting states. Otherwise, infinitely many alternations would give value ∞ to the cost function computed by the automaton.

Thus we define quasi-weak automata in the following way:

▶ **Definition 1.** An alternating B-Buchi automaton is *quasi-weak* if there is a correction function α such that in any play of $\mathcal{A}$ of value $n < \infty$, the number of alternations between accepting and rejecting states is smaller than $\alpha(n)$.

In particular, any weak automaton $\mathcal{A}$ is quasi-weak since we can take $\alpha(n) = k$ for all n, where k is the number of partitions of $\mathcal{A}$. We can also give a structural characterization.

▶ **Proposition 2.** *An alternating B-Büchi automaton is quasi-weak if and only if in any reachable cycle containing both accepting and rejecting states, some counter is incremented but not reset.*

We say a cost function is *quasi-weak* if it is recognized by some quasi-weak B-automaton.

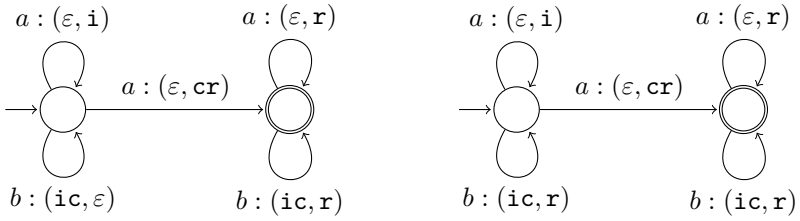
▶ **Proposition 3.** *There exists a cost function over infinite trees which is recognized by a non-deterministic quasi-weak B-automaton, but not by any weak B-automaton. Consequently, quasi-weak B-automata are strictly more expressive than weak B-automata.*

Proof. (Sketch) The idea is to build an explicit cost function f, and for each $n \in \mathbb{N}$ an infinite tree t_n which includes labels that dictate which player controls each position in the game (this is inspired by [1]). These trees are designed such that any alternating B-automaton recognizing f is forced to do $\Theta(n)$ alternations between accepting and rejecting states on t_n. This shows f cannot be computed by a weak B-automaton. On the other hand, we give an explicit non-deterministic quasi-weak B-automaton for f. ◀

3 BS-Automata

We usually work with cost automata with only one type of counter, B or S. In the next section, however, we compose runs from B-Büchi and S-Büchi automata and consequently must work with both counter types simultaneously. We capture this in a *non-deterministic BS-Büchi automaton* $\mathcal{A} = \langle Q, \mathbb{A}, q_0, F_B, F_S, \Gamma_B, \Gamma_S, \Delta \rangle$. Such an automaton defines functions $[\![\mathcal{A}]\!]_B$ and $[\![\mathcal{A}]\!]_S$ as expected (by restricting to one of the counter types).

Let $\mathcal{A}$ and $\mathcal{A}'$ be the following non-deterministic BS-automata on infinite words over $\mathbb{A} := \{a, b, c\}$, each with one B- and one S-counter. We write $a : (d, d')$ if on input a, the output is action d (resp. d') for the B (resp. S) counter. We omit self-loops $c : (\varepsilon, \varepsilon)$.



These automata are very similar. For instance, $[\![\mathcal{A}]\!]_B = [\![\mathcal{A}']\!]_B = |\cdot|_b$. The key difference is $\mathcal{A}'$ is *hierarchical*, with the B-counter above the S-counter. Formally, the counters $\Gamma_B \uplus \Gamma_S$ are globally numbered $[1, k]$ (for $k = |\Gamma_B| + |\Gamma_S|$) and for any action on $\mathbb{B}^{\Gamma_B} \times \mathbb{S}^{\Gamma_S}$ there is some $i \in [1, k]$ such that ε is performed on all counters $j > i$ and $\mathtt{r}$ on all counters $j < i$.

Notice that we have $[\![\mathcal{A}]\!]_S \approx |\cdot|_a$ (if there are a finite number of a's, then the best run of $\mathcal{A}$ moves to the accepting state when reading the final a; otherwise, for every n, there is an accepting run of $\mathcal{A}$ such that the S-counter has value n). In $\mathcal{A}'$, however, the B-counter is higher than the S-counter so $\mathcal{A}'$ forces a reset of the S-counter when a b is read in the initial state. Since there is no a priori bound on the number of b's in the input, this means $[\![\mathcal{A}']\!]_S \not\approx [\![\mathcal{A}]\!]_S$. However, for any fixed m and any u such that $[\![\mathcal{A}]\!]_B(u) \leq m$, the S-value of $\mathcal{A}$ on u is $\approx_{\beta_m}$-equivalent to $\mathcal{A}'$ on u with $\beta_m(n) = n(m+1)$.

This motivates a new equivalence relation $\cong$ which we call *BS-equivalence*. We define $\mathcal{A} \cong \mathcal{A}'$ to hold if there is a correction function α such that (i) $[\![\mathcal{A}]\!]_B \approx_\alpha [\![\mathcal{A}']\!]_B$ and (ii) for any m, there is a correction function β_m such that the S-values of $\mathcal{A}$ and $\mathcal{A}'$ are $\approx_{\beta_m}$-equivalent when restricted to inputs with B-values at most $\alpha(m)$. Although it is technical, this definition captures the notion that two BS-Büchi automata behave in a similar fashion (as in the example above).

It turns out that given any BS-automaton like $\mathcal{A}$, there is an hBS-automaton $\mathcal{A}'$ satisfying $\mathcal{A} \cong \mathcal{A}'$. Moreover, this translation can be done effectively by transducers which read an infinite word of non-hierarchical counter actions and output hierarchical counter

actions. This is in analogy to the deterministic transducer which can be used to translate a Muller condition to a parity condition in the classical setting, or the transducer defined in [6] which translates B-actions to hierarchical B-actions. A similar idea is also used in [2] for automata with both B- and S-counters but in a setting where only boolean properties about boundedness and unboundedness are considered (unlike the quantitative setting here).

▶ **Theorem 4.** *For all sets Γ_B, Γ_S of counters, there exists effectively a history-deterministic hBS-automaton $\mathcal{H}(\Gamma_B, \Gamma_S)$ on infinite words over $\mathbb{B}^{|\Gamma_B|} \times \mathbb{S}^{|\Gamma_S|}$ with $\mathcal{H}(\Gamma_B, \Gamma_S) \approxeq \mathcal{G}(\Gamma_B, \Gamma_S)$ where $\mathcal{G}(\Gamma_B, \Gamma_S)$ is the BS-automaton which copies the counter actions from the input.*

The transducer $\mathcal{H}(\Gamma_B, \Gamma_S)$ has the same set of B counters, but extra copies of the S-counters. The principle of the automaton is to split the input word into sequences of S-actions from $\{\mathtt{i}, \varepsilon\}^*$ which are between resets of the B-counters. It uses one copy of the S-counter to count the number of S-increments within each sequence, and another copy to count the sequences with at least one S-increment. If the S-value is high compared to the B-value, then the transducer will also have a high S-value, obtained from one of the copies.

These transducers are *history-deterministic*, a weakening of traditional determinism [3]. The entire history of the input and the current state are required to determine the next transition (rather than just the current state and input letter). Because the choice of the transition depends only on the past, for any two input words the automaton can find good moves which do not conflict on any common prefix. This means these automata (like deterministic automata) compose well with alternating automata and games: they can be simulated on each play in a game while preserving the value up to $\approx$ or $\approxeq$ (see [3] for more information).

This means that we can use the transducers to transform arbitrary BS-automata over words or trees into hierarchical BS-automata which are easier to work with.

4 Characterization of Quasi-Weak Cost Automata

In this section we prove a Rabin-style characterization for quasi-weak B-automata:

▶ **Theorem 5.** *A cost function f over infinite trees is recognizable by some quasi-weak B-automaton $\mathcal{B}$ if and only if there is a non-deterministic B-Büchi automaton $\mathcal{U}$ and non-deterministic S-Büchi automaton $\mathcal{U}'$ such that $f \approx [\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$.*

The first direction is described in Lemmas 6 and 7 in Section 4.1. The other direction is described in Sections 4.2–4.4, culminating in Theorem 10.

4.1 Simulation

We start by showing that a quasi-weak B-automaton (in fact, any alternating B-Büchi automaton) $\mathcal{A}$ can be simulated by a non-deterministic B-Büchi version.

▶ **Lemma 6.** *Given an alternating B-automaton $\mathcal{B}$, a non-deterministic B-Büchi automaton $\mathcal{U}$ can be effectively constructed such that $[\![\mathcal{B}]\!]_B \approx [\![\mathcal{U}]\!]_B$.*

Proof. (Sketch) In a B-Büchi game, the value of a strategy is the max over all plays compatible with it. Hence, we first show there is a history-deterministic B-Büchi automaton $\mathcal{D}_{\max}$ recognizing $\mathit{max\text{-}play}(w_\sigma^\tau) = \sup\{\mathit{val}(\pi) : \pi \text{ is compatible with } \sigma \text{ and stays on } \tau\}$ on words w_σ^τ which describe the set of plays from a strategy σ which stay on a branch τ.

On input t, the non-deterministic B-Büchi $\mathcal{U}$ guesses a tree t_σ (an annotated version of t over an extended alphabet), checks that the annotations describe a valid finite-memory

strategy σ in $(\mathcal{B}, t)$, and simulates $\mathcal{D}_{\max}$ on each branch in t_σ in order to calculate the value of the strategy (possible since $\mathcal{D}_{\max}$ is history-deterministic). Because non-determinism resolves into taking an infimum, $\mathcal{U}$ calculates the infimum over the values of all finite-memory strategies in $(\mathcal{B}, t)$. Although finite-memory strategies might not achieve the optimal value, they do achieve an $\approx$-equivalent value in B-Büchi games by [13]. Hence, $[\![\mathcal{B}]\!] \approx [\![\mathcal{U}]\!]$. ◀

Proving that $\mathcal{B}$ can be simulated by a non-deterministic S-Büchi automaton $\mathcal{U}'$ is more technical and uses the fact that $\mathcal{B}$ is quasi-weak.

▶ **Lemma 7.** *Given a quasi-weak B-automaton $\mathcal{B}$, a non-deterministic S-Büchi automaton $\mathcal{U}'$ can be effectively constructed such that $[\![\mathcal{B}]\!]_B \approx [\![\mathcal{U}']\!]_S$.*

Proof. (Sketch) The automaton $\mathcal{U}'$ can no longer guess a strategy in $(\mathcal{B}, t)$, since the value of $(\mathcal{B}, t)$ is the infimum over all strategies and non-determinism in an S-automaton resolves into taking a supremum. Instead, we consider a dual game $\overline{(\mathcal{B}, t)}$ where the roles of the players are reversed so Eve tries to maximize the B-value across all strategies. We show there is a history-deterministic S-Büchi automaton $\mathcal{D}_{\min}$ which computes the minimum value of a set of plays from such a game, and show these games admit finite-memory strategies. The S-Büchi automaton $\mathcal{U}'$ guesses a finite-memory strategy in such a game and then simulates $\mathcal{D}_{\min}$ on each branch of the tree annotated with this strategy in order to compute its value. ◀

These simulation lemmas and [13, Lemma 1] imply a new decidability result (extending the class of cost functions over infinite trees for which decidability of $\approx$ is known).

▶ **Corollary 8.** *If f, g are cost functions over infinite trees which are given by quasi-weak B-automata then it is decidable whether or not $f \preccurlyeq g$.*

4.2 Construction from Kupferman and Vardi

We now turn to the other direction of Theorem 5. The corresponding classical result states that given non-deterministic Büchi automata $\mathcal{U}$ and $\mathcal{U}'$ such that $L(\mathcal{U})$ is the complement of $L(\mathcal{U}')$, there is a weak automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{U})$ [9].

The proofs in [12, 9] begin with an analysis of composed runs of $\mathcal{U}$ and $\mathcal{U}'$. Let $m := |Q| \cdot |Q'|$. A *frontier* E is a set of nodes of t such that for any branch π of t, $E \cap \pi$ is a singleton. Kupferman and Vardi [9] define a *trap* for $\mathcal{U}$ and $\mathcal{U}'$ to be a strictly increasing sequence of frontiers $E_0 = \{\epsilon\}, E_1 \ldots, E_m$ such that there exists a tree t, a run R of $\mathcal{U}$ on t, and a run R' of $\mathcal{U}'$ on t satisfying the following properties: for all $0 \leq i < m$ and for all branches π in t, there exists $x, x' \in [e_i^\pi, e_{i+1}^\pi)$ such that $R(x) \in F$ and $R'(x') \in F'$ where $e_0^\pi < \cdots < e_m^\pi$ is the set of nodes from $E_0, \ldots, E_m$ induced by π. The set of positions $[e_i^\pi, e_{i+1}^\pi)$ can be viewed as a block, and each block in a trap witnesses an accepting state from $\mathcal{U}$ and $\mathcal{U}'$.

This is called a trap because $L(\mathcal{U}')$ is the complement of $L(\mathcal{U})$, but a trap implies $L(\mathcal{U}) \cap L(\mathcal{U}') \neq \emptyset$ (using a pumping argument on blocks). The weak automaton $\mathcal{A}$ has Eve (resp. Adam) select a run of $\mathcal{U}$ (resp. $\mathcal{U}'$). The acceptance condition requires that any time an accepting state from $\mathcal{U}'$ is seen, an accepting state from $\mathcal{U}$ is eventually seen. Because of the trap condition, these accepting blocks only need to be counted up to m times (so $\mathcal{A}$ is weak).

4.3 Cost Traps

Now let $\mathcal{U} = \langle Q_{\mathcal{U}}, \mathbb{A}, q_0^{\mathcal{U}}, F_B^{\mathcal{U}}, \Gamma_B^{\mathcal{U}}, \Delta_{\mathcal{U}} \rangle$ (respectively, $\mathcal{U}' = \langle Q_{\mathcal{U}'}, \mathbb{A}, q_0^{\mathcal{U}'}, F_S^{\mathcal{U}'}, \Gamma_S^{\mathcal{U}'}, \Delta_{\mathcal{U}'} \rangle$) be a non-deterministic B-Büchi (respectively, S-Büchi) automaton such that $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$. Our goal is to construct a quasi-weak B-automaton $\mathcal{B}$ which is equivalent to $\mathcal{U}$.

We want to extend the classical case to cost functions, so we seek a notion of "cost trap", which will imply a contradiction with $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$. More specifically, we want a notion of blocks and traps which will witness a bounded B-value from $\mathcal{U}$ on some set of trees but an unbounded S-value for $\mathcal{U}'$ on the same set (showing $[\![\mathcal{U}']\!]_S \not\preccurlyeq [\![\mathcal{U}]\!]_B$). The definition of a block when using arbitrary B- and S-counter actions coming from $\mathcal{U}$ and $\mathcal{U}'$ would be very intricate because it would have to deal with the interaction of the B- and S-actions. In order to avoid this, we switch to working with a non-deterministic hBS-Büchi automaton $\mathcal{A} = \langle Q_{\mathcal{A}}, \mathbb{A}, q_0^{\mathcal{A}}, F_B, F_S, \Gamma_B, \Gamma_S, \delta_{\mathcal{A}} \rangle$ which is BS-equivalent to $\mathcal{U} \times \mathcal{U}' = \langle Q_{\mathcal{U}} \times Q_{\mathcal{U}'}, \mathbb{A}, (q_0^{\mathcal{U}}, q_0^{\mathcal{U}'}), F_B^{\mathcal{U}}, F_S^{\mathcal{U}'}, \Gamma_B^{\mathcal{U}}, \Gamma_S^{\mathcal{U}'}, \Delta_{\mathcal{U}\times\mathcal{U}'} \rangle$ but uses hierarchical counters.

A *block* based on hierarchical BS-actions from $\mathcal{A}$ has accepting states from both F_B and F_S (corresponding to accepting states for $\mathcal{U}$ and $\mathcal{U}'$), but it also has a reset for B-counter γ if γ is incremented in that block (in order to ensure pumping does not inflate the B-value). The number of blocks required is also increased to $m := (|Q_{\mathcal{A}}| + 2)^{|\Gamma_S|+1}$ for technical reasons.

A *cost trap* for $\mathcal{A}$ is a frontier E_m and for every branch π up to E_m a strictly increasing set of nodes $e_0^\pi < \cdots < e_m^\pi \in E_m$ such that there exists a tree t and a run R of $\mathcal{A}$ on t with $val_S(R) > |Q_{\mathcal{A}}|$ satisfying the following properties: for all $0 \leq i < m$ and for all branches π, $[e_i^\pi, e_{i+1}^\pi)$ is a block; if branches π_1 and π_2 share some prefix up to position y and $x < y$ is the first position with $e_i^{\pi_1} = x$ and $e_i^{\pi_2} \neq x$ then $e_i^{\pi_2} > y$ (i.e. pumping blocks from π_2 does not damage blocks from π_1).

A pumping argument shows a cost trap implies $\mathcal{U}$ and $\mathcal{U}'$ are not equivalent.

▶ **Proposition 9.** *Let $\mathcal{U}$ (respectively, $\mathcal{U}'$) be non-deterministic B-Büchi (respectively, S-Büchi). Let $\mathcal{A} \cong \mathcal{U} \times \mathcal{U}'$ be a non-deterministic hBS-automaton. If there exists a cost trap for $\mathcal{A}$, then $[\![\mathcal{U}']\!] \not\preccurlyeq [\![\mathcal{U}]\!]$.*

4.4 Construction of Quasi-Weak B-Automaton $\mathcal{B}$

Given $\mathcal{U}$ and $\mathcal{U}'$ with $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$, we can effectively build a quasi-weak B-automaton $\mathcal{B}$ which on an input tree t,

- simulates in parallel $\mathcal{U}$ (driven by Eve) and $\mathcal{U}'$ (driven by Adam) over t;
- runs the hBS-transducer $\mathcal{H}(\Gamma_B^{\mathcal{U}}, \Gamma_S^{\mathcal{U}'})$ over the composed actions from $\mathcal{U}$ and $\mathcal{U}'$;
- analyzes the output of this transducer together with the accepting states of $\mathcal{U}$ and $\mathcal{U}'$, keeping track of blocks (see below);
- outputs the B-actions of $\mathcal{U}$.

The key difference from the classical case is in the block counting. In [9], the block number only increases and it suffices to count up to a fixed bound. Since each block contains at most 2 alternations between accepting and rejecting states, this results in a weak automaton.

Here, we also have to forbid in any block the presence of an increment for some counter γ without a reset for γ. However, it may be the case that on a branch of a run of $\mathcal{U}$ some counter is incremented but is never reset. So the automaton $\mathcal{B}$ may start counting blocks only to have to restart the counting if an increment is seen which does not have a later reset. But this means that any decrease in the block number corresponds to an increase in the

cost of the play. Hence, the bound on the number of alternations depends on the value of the automaton, which is exactly the property of a quasi-weak automaton.

The idea for the proof that $[\![\mathcal{B}]\!]_B \approx [\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$ is that if $\mathcal{U}$ accepts some t with low value, then it gives Eve a strategy of the same value in $(\mathcal{B}, t)$. On the other hand, assuming (for the sake of contradiction) that Eve has a low-value strategy in $(\mathcal{B}, t)$ but $\mathcal{U}$ actually assigns t a high value results in a cost trap, which is absurd. Hence, we get the main result:

▶ **Theorem 10.** *If there is a non-deterministic B-Büchi automaton $\mathcal{U}$ and non-deterministic S-Büchi automaton $\mathcal{U}'$ such that $[\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$, then we can effectively construct a quasi-weak alternating B-automaton $\mathcal{B}$ such that $[\![\mathcal{B}]\!]_B \approx [\![\mathcal{U}]\!]_B \approx [\![\mathcal{U}']\!]_S$.*

We remark that when restricted to languages, this corresponds to the result from [9] since (i) if there are non-deterministic Büchi automata $\mathcal{U}$ and $\mathcal{U}'$ (without counters) recognizing a language and its complement, respectively, then $[\![\mathcal{U}]\!]_B = [\![\mathcal{U}']\!]_S$ and (ii) quasi-weak and weak automata coincide when the automata have no counters.

5 Conclusion

We have introduced quasi-weak cost automata as a variant of weak automata which uses the counters to bound the number of alternations between accepting and rejecting states. We have shown quasi-weak cost automata are strictly more expressive than weak cost automata over infinite trees. Moreover, it is the quasi-weak class of automata, rather than the more traditional weak cost automata, which admits a Rabin-style characterization with non-deterministic B-Büchi and S-Büchi automata. The question of a characterization for weak cost automata over infinite trees remains open (it would likely involve some further restrictions on the actions of the counters in the non-deterministic B-Büchi and S-Büchi automata).

Combined with results from [13], our Rabin-style characterization of quasi-weak automata implies the decidability of $f \preccurlyeq g$ and $f \approx g$ when f, g are defined by quasi-weak B-automata. Consequently, this work extends the class of cost functions over infinite trees for which $\approx$ is known to be decidable. Deciding $\preccurlyeq$ and $\approx$ for all regular cost functions over infinite trees remains a challenging open problem which would imply (by [5]) the decidability of the parity-index problem.

Finally, it was known from [13] that weak cost automata and cost WMSO are equivalent. The logic side of quasi-weak cost automata remains to be explored in future work.

Acknowledgements We are grateful to Thomas Colcombet for having made this joint work possible, and for many helpful discussions.

References

1 André Arnold and Damian Niwinski. Continuous separation of game languages. *Fundam. Inform.*, 81(1-3):19–28, 2007.

2 Mikolaj Bojanczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS*, pages 285–296. IEEE Computer Society, 2006.

3 Thomas Colcombet. The Theory of Stabilisation Monoids and Regular Cost Functions. In *ICALP (2)*, volume 5556 of *LNCS*, pages 139–150. Springer, 2009.

4 Thomas Colcombet and Christof Löding. The nesting-depth of disjunctive mu-calculus. In Michael Kaminski and Simone Martini, editors, *CSL*, volume 5213 of *LNCS*, pages 416–430. Springer, 2008.

5 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In Luca Aceto, Ivan Damgard, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *LNCS*, pages 398–409. Springer, 2008.

6 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79. IEEE Computer Society, 2010.

7 Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982.

8 Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO - Theoretical Informatics and Applications*, 39(3):455–509, 2005.

9 Orna Kupferman and Moshe Y. Vardi. The weakness of self-complementation. In Christoph Meinel and Sophie Tison, editors, *STACS*, volume 1563 of *LNCS*, pages 455–466. Springer, 1999.

10 David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata. The weak monadic theory of the tree, and its complexity. In Laurent Kott, editor, *ICALP*, volume 226 of *LNCS*, pages 275–283. Springer, 1986.

11 Damian Niwinski and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Electr. Notes Theor. Comput. Sci.*, 123:195–208, 2005.

12 Michael O. Rabin. Weakly definable relations and special automata. In *Mathematical Logic and Foundations of Set Theory (Proc. Internat. Colloq., Jerusalem, 1968)*, pages 1–23. North-Holland, Amsterdam, 1970.

13 Michael Vanden Boom. Weak cost monadic logic over infinite trees. In Filip Murlak and Piotr Sankowski, editors, *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2011.

Using non-convex approximations for efficient analysis of timed automata

Frédéric Herbreteau[1], Dileep Kini[2], B. Srivathsan[1], and Igor Walukiewicz[1]

1 **Université de Bordeaux, IPB, CNRS, LaBRI UMR5800**
2 **Indian Institute of Technology Bombay, Department of Computer Science and Engineering**

Abstract

The reachability problem for timed automata asks if there exists a path from an initial state to a target state. The standard solution to this problem involves computing the zone graph of the automaton, which in principle could be infinite. In order to make the graph finite, zones are approximated using an extrapolation operator. For reasons of efficiency in current algorithms extrapolation of a zone is always a zone; and in particular it is convex.

In this paper, we propose to solve the reachability problem without such extrapolation operators. To ensure termination, we provide an efficient algorithm to check if a zone is included in the so called region closure of another. Although theoretically better, closure cannot be used in the standard algorithm since a closure of a zone may not be convex.

An additional benefit of the proposed approach is that it permits to calculate approximating parameters on-the-fly during exploration of the zone graph, as opposed to the current methods which do it by a static analysis of the automaton prior to the exploration. This allows for further improvements in the algorithm. Promising experimental results are presented.

1998 ACM Subject Classification D.2.4 Software/Program Verification; F.3.1 Specifying and Verifying and Reasoning about Programs; F.4.1 Mathematical Logic

Keywords and phrases Timed Automata; Model-checking; Non-convex abstraction; On-the-fly abstraction

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.78

1 Introduction

Timed automata [1] are obtained from finite automata by adding clocks that can be reset and whose values can be compared with constants. The crucial property of timed automata is that their reachability problem is decidable: one can check if a given target state is reachable from the initial state. Reachability algorithms are at the core of verification tools like Uppaal [4] or RED [16], and are used in industrial case studies [11, 6]. The standard solution constructs a search tree whose nodes are approximations of zones. In this paper we give an efficient algorithm for checking if a zone is included in an approximation of another zone. This enables a reachability algorithm to work with search trees whose nodes are just unapproximated zones. This has numerous advantages: one can use non-convex approximations, and one can compute approximating parameters on the fly.

The first solution to the reachability problem has used *regions*, which are equivalence classes of clock valuations. Subsequent research has shown that the region abstraction is very inefficient and an other method using *zones* instead of regions has been proposed. This

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 78–89

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

can be implemented efficiently using DBMs [10] and is used at present in almost all timed-verification tools. The number of reachable zones can be infinite, so one needs an abstraction operator to get a finite approximation. The simplest is to approximate a zone with the set of regions it intersects, the so called *closure* of a zone. Unfortunately, the closure may not always be convex and no efficient representation of closures is known. For this reason implementations use another convex approximation that is also based on (refined) regions.

We propose a new algorithm for the reachability problem using closures of zones. To this effect we provide an efficient algorithm for checking whether a zone is included in a closure of another zone. In consequence we can work with non-convex approximations without a need to store them explicitly.

Thresholds for approximations are very important for efficient implementation. Good thresholds give substantial gains in time and space. The simplest approach is to take as a threshold the maximal constant appearing in a transition of the automaton. A considerable gain in efficiency can be obtained by analyzing the graph of the automaton and calculating thresholds specific for each clock and state of the automaton [2]. An even more efficient approach is the so called LU-approximation that distinguishes between upper and lower bounds [3]. This is the method used in the current implementation of UPPAAL. We show that we can accommodate closure on top of the LU-approximation at no extra cost.

Since our algorithm never stores approximations, we can compute thresholds on-the-fly. This means that our computation of thresholds does not take into account unreachable states. In consequence in some cases we get much better LU-thresholds than those obtained by static analysis. This happens in particular in a very common context of analysis of parallel compositions of timed automata.

Related work

The topic of this paper is approximation of zones and efficient handling of them. We show that it is possible to use non-convex approximations and that it can be done efficiently. In particular, we improve on state of the art approximations [3]. Every forward algorithm needs approximations, so our work can apply to tools like RED or UPPAAL.

Recent work [15] reports on backward analysis approach using general linear constraints. This approach does not use approximations and relies on SMT solver to simplify the constraints. Comparing forward and backward methods would require a substantial test suite, and is not the subject of this paper.

Organization of the paper

The next section presents the basic notions and recalls some of their properties. Section 3 describes the new algorithm for efficient inclusion test between a zone and a closure of another zone. The algorithm constructing the search tree and calculating approximations on-the-fly is presented in Section 4. Some results obtained with a prototype implementation are presented in the last section. All missing proofs are presented in the full version of the paper [13].

2 Preliminaries

2.1 Timed automata and the reachability problem

Let X be a set of clocks, i.e., variables that range over $\mathbb{R}_{\geq 0}$, the set of non-negative real numbers. A *clock constraint* is a conjunction of constraints $x\#c$ for $x \in X$, $\# \in \{<, \leq, =$

$, \geq, >\}$ and $c \in \mathbb{N}$, e.g. $(x \leq 3 \wedge y > 0)$. Let $\Phi(X)$ denote the set of clock constraints over clock variables X. A *clock valuation* over X is a function $\nu : X \to \mathbb{R}_{\geq 0}$. We denote $\mathbb{R}^X_{\geq 0}$ the set of clock valuations over X, and $\mathbf{0}$ the valuation that associates 0 to every clock in X. We write $\nu \vDash \phi$ when ν satisfies $\phi \in \Phi(X)$, i.e. when every constraint in ϕ holds after replacing every x by $\nu(x)$. For $\delta \in \mathbb{R}_{\geq 0}$, let $\nu + \delta$ be the valuation that associates $\nu(x) + \delta$ to every clock x. For $R \subseteq X$, let $[R]\nu$ be the valuation that sets x to 0 if $x \in R$, and that sets x to $\nu(x)$ otherwise.

A *Timed Automaton (TA)* is a tuple $\mathcal{A} = (Q, q_0, X, T, Acc)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, X is a finite set of clocks, $Acc \subseteq Q$ is a set of accepting states, and $T \subseteq Q \times \Phi(X) \times 2^X \times Q$ is a finite set of transitions (q, g, R, q') where g is a *guard*, and R is the set of clocks that are *reset* on the transition. An example of a TA is depicted in Figure 1. The class of TA we consider is commonly known as diagonal-free TA since clock comparisons like $x - y \leq 1$ are disallowed. Notice that since we are interested in state reachability, considering timed automata without state invariants does not entail any loss of generality. Indeed, state invariants can be added to guards, then removed, while preserving state reachability.

A *configuration* of $\mathcal{A}$ is a pair $(q, \nu) \in Q \times \mathbb{R}^X_{\geq 0}$; $(q_0, \mathbf{0})$ is the *initial configuration*. We write $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$ if there exists $\delta \in \mathbb{R}_{\geq 0}$ and a transition $t = (q, g, R, q')$ in $\mathcal{A}$ such that $\nu + \delta \vDash g$, and $\nu' = [R]\nu$. Then (q', ν') is called a *successor* of (q, ν). A *run* of $\mathcal{A}$ is a finite sequence of transitions: $(q_0, \nu_0) \xrightarrow{\delta_0, t_0} (q_1, \nu_1) \xrightarrow{\delta_1, t_1} \cdots (q_n, \nu_n)$ starting from $(q_0, \nu_0) = (q_0, \mathbf{0})$.

A run is *accepting* if it ends in a configuration (q_n, ν_n) with $q_n \in Acc$. The *reachability problem* is to decide whether a given automaton has an accepting run. This problem is known to be PSPACE-complete [1, 8].

2.2 Symbolic semantics for timed automata

The reachability problem is solved using so-called symbolic semantics. It considers sets of (uncountably many) valuations instead of valuations separately. A *zone* is a set of valuations defined by a conjunction of two kinds of constraints: comparison of difference between two clocks with an integer like $x - y \# c$, or comparison of a single clock with an integer like $x \# c$, where $\# \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. For instance $(x - y \geq 1) \wedge (y < 2)$ is a zone. The transition relation on valuations is transferred to zones as follows. We have $(q, Z) \xrightarrow{t} (q', Z')$ if Z' is the set of valuations ν' such that $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$ for some $\nu \in Z$ and $\delta \in \mathbb{R}_{\geq 0}$. The node (q', Z') is called a successor of (q, Z). It can be checked that if Z is a zone, then Z' is also a zone.

The *zone graph* of $\mathcal{A}$, denoted $ZG(\mathcal{A})$, has nodes of the form (q, Z) with initial node $(q_0, \{\mathbf{0}\})$, and edges defined as above. Immediately from the definition of $ZG(\mathcal{A})$ we infer that $\mathcal{A}$ has an accepting run iff there is a node (q, Z) reachable in $ZG(\mathcal{A})$ with $q \in Acc$.

Now, every node (q, Z) has finitely many successors: at most one successor of (q, Z) per transition in $\mathcal{A}$. Still a reachability algorithm may not terminate as the number of reachable nodes in $ZG(\mathcal{A})$ may not be finite [9]. The next step is thus to define an abstract semantics of $\mathcal{A}$ as a finite graph. The basic idea is to define a finite partition of the set of valuations $\mathbb{R}^X_{\geq 0}$. Then, instead of considering nodes (q, S) with set of valuations S (e.g. zones Z), one considers a union of the parts of $\mathbb{R}^X_{\geq 0}$ that intersect S. This gives the finite abstraction.

Let us consider a *bound function* associating to each clock x of $\mathcal{A}$ a bound $\alpha_x \in \mathbb{N}$. A *region* [1] with respect to α is the set of valuations specified as follows:

1. for each clock $x \in X$, one constraint from the set:

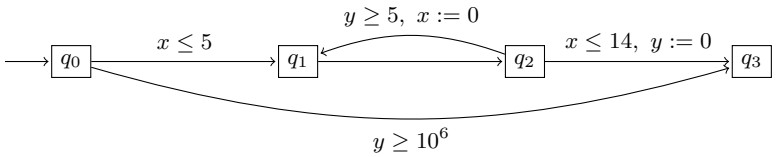


Figure 1 Timed automaton $\mathcal{A}$.

$$\{x = c \mid c = 0, \ldots, \alpha_x\} \cup \{c - 1 < x < c \mid c = 1, \ldots, \alpha_x\} \cup \{x > \alpha_x\}$$

2. for each pair of clocks x, y having interval constraints: $c - 1 < x < c$ and $d - 1 < y < d$, it is specified if $fract(x)$ is less than, equal to or greater than $fract(y)$.

It can be checked that the set of regions is a finite partition of $\mathbb{R}^X_{\geq 0}$.

The *closure abstraction* of a set of valuations S, denoted $Closure_\alpha(S)$, is the union of the regions that intersect S [7]. A simulation graph, denoted $SG_\alpha(\mathcal{A})$, has nodes of the form (q, S) where q is a state of $\mathcal{A}$ and $S \subseteq \mathbb{R}^X_{\geq 0}$ is a set of valuations. The initial node of $SG_\alpha(\mathcal{A})$ is $(q_0, \{\mathbf{0}\})$. There is an edge $(q, S) \xrightarrow{t} (q', Closure_\alpha(S'))$ in $SG_\alpha(\mathcal{A})$ iff S' is the set of valuations ν' such that $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$ for some $\nu \in S$ and $\delta \in \mathbb{R}_{\geq 0}$. Notice that the reachable part of $SG_\alpha(\mathcal{A})$ is finite since the number of regions is finite.

The definition of the graph $SG_\alpha(\mathcal{A})$ is parametrized by a bound function α. It is well-known that if we take $\alpha_\mathcal{A}$ associating to each clock x the maximal integer c such that $x \# c$ appears in some guard of $\mathcal{A}$ then $SG_\alpha(\mathcal{A})$ preserves the reachability properties.

▶ **Theorem 1.** *[7] $\mathcal{A}$ has an accepting run iff there is a reachable node (q, S) in $SG_\alpha(\mathcal{A})$ with $q \in Acc$ and $\alpha_\mathcal{A} \leq \alpha$.*

For efficiency it is important to have a good bound function α. The nodes of $SG_\alpha(\mathcal{A})$ are unions of regions. Hence the size of $SG_\alpha(\mathcal{A})$ depends on the number of regions which is $\mathcal{O}\big(|X|!.2^{|X|}.\prod_{x \in X}(2.\alpha_x + 2)\big)$ [1]. It follows that smaller values for α yield a coarser, hence smaller, symbolic graph $SG_\alpha(\mathcal{A})$. Note that current implementations do not use closure but some convex under-approximation of it that makes the graph even bigger.

It has been observed in [2] that instead of considering a global bound function $\alpha_\mathcal{A}$ for all states in $\mathcal{A}$, one can use different functions in each state of the automaton. Consider for instance the automaton $\mathcal{A}$ in Figure 1. Looking at the guards, we get that $\alpha_x = 14$ and $\alpha_y = 10^6$. Yet, a closer look at the automaton reveals that in the state q_2 it is enough to take the bound $\alpha_y(q_2) = 5$. This observation from [2] points out that one can often get very big gains by associating a bound function $\alpha(q)$ to each state q in $\mathcal{A}$ that is later used for the abstraction of nodes of the form $(q, Closure_{\alpha(q)}(S))$. In op. cit. an algorithm for inferring bounds based on static analysis of the structure of the automaton is proposed. In Section 4.2 we will show how to calculate these bounds on-the-fly during the exploration of the automaton's state space.

3 Efficient testing of inclusion in a closure of a zone

The tests of the form $Z \subseteq Closure_\alpha(Z')$ will be at the core of the new algorithm we propose. This is an important difference with respect to the standard algorithm that makes the tests of the form $Z \subseteq Z'$. The latter tests are done in $\mathcal{O}(|X|^2)$ time, where $|X|$ is the number of clocks. We present in this section a simple algorithm that can do the tests $Z \subseteq Closure_\alpha(Z')$ at the same complexity with neither the need to represent nor to compute the closure.

We start by examining the question as to how one decides if a region R intersects a zone Z. The important point is that it is enough to verify that the projection on every pair of variables is nonempty. This is the cornerstone for the efficient inclusion testing algorithm that even extends to LU-approximations.

3.1 When is $R \cap Z$ empty

It will be very convenient to represent zones by *distance graphs*. Such a graph has clocks as vertices, with an additional special clock x_0 representing constant 0. For readability, we will often write 0 instead of x_0. Between every two vertices there is an edge with a weight of the form $(\preccurlyeq, c)$ where $c \in \mathbb{Z} \cup \{\infty\}$ and $\preccurlyeq$ is either $\leq$ or $<$. An edge $x \xrightarrow{\preccurlyeq c} y$ represents a constraint $y - x \preccurlyeq c$: or in words, the distance from x to y is bounded by c. Let $[\![G]\!]$ be the set of valuations of clock variables satisfying all the constraints given by the edges of G with the restriction that the value of x_0 is 0.

An arithmetic over the weights $(\preccurlyeq, c)$ can be defined as follows [5].
Equality $(\preccurlyeq_1, c_1) = (\preccurlyeq_2, c_2)$ if $c_1 = c_2$ and $\preccurlyeq_1 = \preccurlyeq_2$.
Addition $(\preccurlyeq_1, c_1) + (\preccurlyeq_2, c_2) = (\preccurlyeq, c_1 + c_2)$ where $\preccurlyeq = <$ iff either $\preccurlyeq_1$ or $\preccurlyeq_2$ is $<$.
Minus $-(\preccurlyeq, c) = (\preccurlyeq, -c)$.
Order $(\preccurlyeq_1, c_1) < (\preccurlyeq_2, c_2)$ if either $c_1 < c_2$ or ($c_1 = c_2$ and $\preccurlyeq_1 = <$ and $\preccurlyeq_2 = \leq$).
Floor $\lfloor (<, c) \rfloor = (\leq, c-1)$ and $\lfloor (\leq, c) \rfloor = (\leq, c)$.
This arithmetic lets us talk about the weight of a path as a weight of the sum of its edges. A cycle in a distance graph G is said to be *negative* if the sum of the weights of its edges is at most $(<, 0)$; otherwise the cycle is *positive*. The following useful proposition is folklore.

▶ **Proposition 2.** *A distance graph G has only positive cycles iff $[\![G]\!] \neq \emptyset$.*

A distance graph is in *canonical form* if the weight of the edge from x to y is the lower bound of the weights of paths from x to y. A *distance graph of a region* R, denoted G_R, is the canonical graph representing all the constraints defining R. Similarly G_Z for a zone Z.

We can now state a necessary and sufficient condition for the intersection $R \cap Z$ to be empty in terms of cycles in distance graphs. We denote by R_{xy} the weight of the edge $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ in the canonical distance graph representing R. Similarly for Z.

▶ **Proposition 3.** *Let R be a region and let Z be a zone. The intersection $R \cap Z$ is empty iff there exist variables x, y such that $Z_{yx} + R_{xy} \leq (<, 0)$.*

A variant of this fact has been proven as an intermediate step of Proposition 2 in [7].

3.2 Efficient inclusion testing

Our goal is to efficiently perform the test $Z \subseteq \mathit{Closure}(Z')$ for two zones Z and Z'. We are aiming at $\mathcal{O}(|X|^2)$ complexity, since this is the complexity of current algorithms used for checking inclusion of two zones. Proposition 3 can be used to efficiently test the inclusion $R \subseteq \mathit{Closure}(Z')$. It remains to understand what are the regions intersecting the zone Z and then to consider all possible cases. The next lemma basically says that every consistent instantiation of an edge in G_Z leads to a region intersecting Z.

▶ **Lemma 4.** *Let G be a distance graph in canonical form, with all cycles positive. Let x, y be two variables, and let $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ and $y \xrightarrow{\preccurlyeq_{yx} c_{yx}} x$ be edges in G. For every $d \in \mathbb{R}$ such that $d \preccurlyeq_{xy} c_{xy}$ and $-d \preccurlyeq_{yx} c_{yx}$ there exists a valuation $v \in [\![G]\!]$ with $v(y) - v(x) = d$.*

Thanks to this lemma it is enough to look at edges of G_Z one by one to see what regions we can get. This insight is used to get the desired efficient inclusion test

▶ **Theorem 5.** *Let Z, Z' be zones. Then, $Z \not\subseteq Closure_\alpha(Z')$ iff there exist variables x, y, both different from x_0, such that one of the following conditions hold:*

1. $Z'_{0x} < Z_{0x}$ *and* $Z'_{0x} \leq (\leq, \alpha_x)$, *or*
2. $Z'_{x0} < Z_{x0}$ *and* $Z_{x0} \geq (\leq, -\alpha_x)$, *or*
3. $Z_{x0} \geq (\leq, -\alpha_x)$ *and* $Z'_{xy} < Z_{xy}$ *and* $Z'_{xy} \leq (\leq, \alpha_y) + \lfloor Z_{x0} \rfloor$.

Comparison with the algorithm for $Z \subseteq Z'$

Given two zones Z and Z', the procedure for checking $Z \subseteq Z'$ works on two graphs G_Z and $G_{Z'}$ that are in canonical form. This form reduces the inclusion test to comparing the edges of the graphs one by one. Note that our algorithm for $Z \subseteq Closure_\alpha(Z')$ does not do worse. It works on G_Z and $G_{Z'}$ too. The edge by edge checks are only marginally more complicated. The overall procedure is still $\mathcal{O}(|X|^2)$.

3.3 Handling LU-approximation

In [3] the authors propose to distinguish between maximal constants used in upper and lower bounds comparisons: for each clock x, $L_x \in \mathbb{N} \cup \{-\infty\}$ represents the maximal constant c such that there exists a constraint $x > c$ or $x \geq c$ in a guard of a transition in the automaton; dually, $U_x \in \mathbb{N} \cup \{-\infty\}$ represents the maximal constant c such that there is a constraint $x < c$ or $x \leq c$ in a guard of a transition. If such a c does not exist, then it is considered to be $-\infty$. They have introduced an extrapolation operator $Extra^+_{LU}(Z)$ that takes into account this information. This is probably the best presently known convex abstraction of zones.

We now explain how to extend our inclusion test to handle LU approximation, namely given Z and Z' how to directly check $Z \subseteq Closure_\alpha(Extra^+_{LU}(Z'))$ efficiently. Observe that for each x, the maximal constant α_x is the maximum of L_x and U_x. In the sequel, this is denoted $Z \subseteq Closure^+_{LU}(Z')$. For this we need to understand first when a region intersecting Z intersects $Extra^+_{LU}(Z')$. Therefore, we study the conditions that a region R should satisfy if it intersects $Extra^+_{LU}(Z)$ for a zone Z.

We recall the definition given in [3] that has originally been presented using difference bound matrices (DBM). In a DBM $(c_{ij}, \prec_{i,j})$ stands for $x_i - x_j \prec_{i,j} c_{i,j}$. In the language of distance graphs, this corresponds to an edge $x_j \xrightarrow{\prec_{i,j} c_{i,j}} x_i$; hence to Z_{ji} in our notation. Let Z^+ denote $Extra^+_{LU}(Z)$ and G_{Z^+} its distance graph. We have:

$$Z^+_{xy} = \begin{cases} (<, \infty) & \text{if } Z_{xy} > (\leq, L_y) \\ (<, \infty) & \text{if } -Z_{y0} > (\leq, L_y) \\ (<, \infty) & \text{if } -Z_{x0} > (\leq, U_x), y \neq 0 \\ (<, -U_x) & \text{if } -Z_{x0} > (\leq, U_x), y = 0 \\ Z_{xy} & \text{otherwise.} \end{cases} \tag{1}$$

From this definition it will be important for us to note that G_{Z^+} is G_Z with some weights put to $(<, \infty)$ and some weights on the edges to x_0 put to $(<, -U_x)$. Note that $Extra^+_{LU}(Z')$ is not in the canonical form. If we put $Extra^+_{LU}(Z')$ into the canonical form then we could just use Theorem 5. We cannot afford to do this since canonization can take cubic time [5]. The following theorem implies that we can do the test without canonizing $Extra^+_{LU}(Z')$. Hence we can get a simple quadratic test also in this case.

▶ **Theorem 6.** *Let Z, Z' be zones. Let Z'^{+} denote $Extra^{+}_{LU}(Z')$ obtained from Z' using Equation 1 for each edge. Note that Z'^{+} is not necessarily in canonical form. Then, we get that $Z \not\subseteq Closure_{\alpha}(Z'^{+})$ iff there exist variables x, y different form x_0 such that one of the following conditions hold:*

1. $Z'^{+}_{0x} < Z_{0x}$ *and* $Z'^{+}_{0x} \leq (\leq, \alpha_x)$, *or*
2. $Z'^{+}_{x0} < Z_{x0}$ *and* $Z_{x0} \geq (\leq, -\alpha_x)$, *or*
3. $Z_{x0} \geq (\leq, -\alpha_x)$ *and* $Z'^{+}_{xy} < Z_{xy}$ *and* $Z'^{+}_{xy} \leq (\leq, \alpha_y) + \lfloor Z_{x0} \rfloor$.

4 A New Algorithm for Reachability

Our goal is to decide if a final state of a given timed automaton is reachable. We do it by computing a finite prefix of the reachability tree of the zone graph $ZG(\mathcal{A})$ that is sufficient to solve the reachability problem. Finiteness is ensured by not exploring a node (q, Z) if there exists a (q, Z') such that $Z \subseteq Closure_{\alpha}(Z')$, for a suitable α. We will first describe a simple algorithm based on the closure and then we will address the issue of finding tighter bounds for the clock values.

4.1 The basic algorithm

Given a timed automaton $\mathcal{A}$ we first calculate the bound function $\alpha_{\mathcal{A}}$ as described just before Theorem 1. Each node in the tree that we compute is of the form (q, Z), where q is a state of the automaton, and Z is an unapproximated zone. The root node is (q_0, Z_0), which is the initial node of $ZG(\mathcal{A})$. The algorithm performs a depth first search: at a node (q, Z), a transition $t = (q, g, r, q')$ not yet considered for exploration is picked and the successor (q', Z') is computed where $(q, Z) \xrightarrow{t} (q', Z')$ in $ZG(\mathcal{A})$. If q' is a final state and Z' is not empty then the algorithm terminates. Otherwise the search continues from (q', Z') unless there is already a node (q', Z'') with $Z' \subseteq Closure_{\alpha_{\mathcal{A}}}(Z'')$ in the current tree.

The correctness of the algorithm is straightforward. It follows from the fact that if $Z' \subseteq Closure_{\alpha_{\mathcal{A}}}(Z'')$ then all the states reachable from (q', Z') are reachable from (q', Z'') and hence it is not necessary to explore the tree from (q', Z'). Termination of the algorithm is ensured since there are finitely many sets of the form $Closure_{\alpha_{\mathcal{A}}}(Z)$. Indeed, the algorithm will construct a prefix of the reachability tree of $SG_{\alpha}(\mathcal{A})$ as described in Theorem 1.

The above algorithm does not use the classical extrapolation operator named $Extra^{+}_{M}$ in [3] and $Extra^{+}_{\alpha}$ hereafter, but the coarser $Closure_{\alpha}$ operator [7]. This is possible since the algorithm does not need to represent $Closure_{\alpha}(Z)$, which is in general not a zone. Instead of storing $Closure_{\alpha}(Z)$ the algorithm just stores Z and performs tests $Z \subseteq Closure_{\alpha}(Z')$ each time it is needed (in contrast to Algorithm 2 in [7]). This is as efficient as testing $Z \subseteq Z'$ thanks to the algorithm presented in the previous section.

Since $Closure_{\alpha}$ is a coarser abstraction, this simple algorithm already covers some of the optimizations of the standard algorithm. For example the $Extra^{+}_{\alpha}(Z)$ abstraction proposed in [3] is subsumed since $Extra^{+}_{\alpha}(Z) \subseteq Closure_{\alpha}(Z)$ for any zone Z [7, 3]. Other important optimizations of the standard algorithm concern finer computation of bounding functions α. We now show that the structure of the proposed algorithm allows to improve this too.

4.2 Computing clock bounds on-the-fly

We can improve on the idea of Behrmann et al. [2] of computing a bound function α_q for each state q. We will compute these bounding functions on-the-fly and they will depend also on a zone and not just a state. An obvious gain is that we will never consider constraints

Algorithm 1 Reachability algorithm with on-the-fly bound computation and non-convex abstraction.

```
1  function main():
2    push((q0, Z0, α0), stack)
3    while (stack ≠ ∅) do
4      (q, Z, α) := top(stack); pop(stack)
5      explore(q, Z, α)
6      resolve()
7    return "empty"
8
9  function explore(q, Z, α):
10   if (q is accepting)
11     exit "not empty"
12   if (∃ (q, Z', α') nontentative
13             and s.t. Z ⊆ Closure_α'(Z'))
14     mark (q, Z, α) tentative wrt (q, Z', α')
15     α := α'; propagate(parent(q, Z, α))
16   else
17     propagate(q, Z, α)
18     for each (q_s, Z_s, α_s) in children(q, Z, α) do
19       if (Z_s ≠ ∅)
20         explore(q_s, Z_s, α_s)
21
22 function resolve():
23   for each (q, Z, α) tentative wrt (q, Z', α') do
24     if (Z ⊈ Closure_α'(Z'))
25       mark (q, Z, α) nontentative
26       α := −∞; propagate(parent(q, Z, α))
27       push((q, Z, α), stack)
28
29 function propagate(q, Z, α):
30   α := max_{(q,Z,α) --g;R--> (q',Z',α')} maxedge(g, R, α')
31   if (α has changed)
32     for each (q_t, Z_t, α_t) tentative wrt (q, Z, α) do
33       α_t := α; propagate(parent(q_t, Z_t, α_t))
34     if ((q, Z, α) ≠ (q0, Z0, α0))
35       propagate(parent(q, Z, α))
36
37 function maxedge(g, R, α):
38   let α_R = λx. if x ∈ R then −∞ else α(x)
39   let α_g = λx. if x#c in g then c else −∞
40   return (λx. max(α_R(x), α_g(x)))
```

coming from unreachable transitions. We comment more on advantages of this approach in Section 5.

Our modified algorithm is given in Figure 1. It computes a tree whose nodes are triples (q, Z, α) where (q, Z) is a node of $ZG(\mathcal{A})$ and α is a bound function. Each node (q, Z, α) has as many child nodes (q_s, Z_s, α_s) as there are successors (q_s, Z_s) of (q, Z) in $ZG(\mathcal{A})$. Notice that this includes successors with an empty zone Z_s, which are however not further unfolded. These nodes must be included for correctness of our constant propagation procedure. By default bound functions map each clock to $-\infty$. They are later updated as explained below. Each node is further marked either *tentative* or *nontentative*. The leaf nodes (q, Z, α) of the tree are either deadlock nodes (either there is no transition out of state q or Z is empty), or *tentative* nodes. All the other nodes are marked *nontentative*.

Our algorithm starts from the root node (q_0, Z_0, α_0), consisting of the initial state, initial zone, and the function mapping each clock to $-\infty$. It repeatedly alternates an exploration and a resolution phase as described below.

Exploration phase

Before exploring a node $n = (q, Z, \alpha)$ the function `explore` checks if q is accepting and Z is not empty; if it is so then $\mathcal{A}$ has an accepting run. Otherwise the algorithm checks if there exists a *nontentative* node $n' = (q', Z', \alpha')$ in the current tree such that $q = q'$ and $Z \subseteq Closure_{\alpha'}(Z')$. If yes, n becomes a *tentative* node and its exploration is temporarily stopped as each state reachable from n is also reachable from n'. If none of these holds, the successors of the node are explored. The exploration terminates since $Closure_\alpha$ has a finite range.

When the exploration algorithm gets to a new node, it propagates the bounds from this node to all its predecessors. The goal of these propagations is to maintain the following invariant. For every node $n = (q, Z, \alpha)$:

1. if n is *nontentative*, then α is the maximum of the α_s from all successor nodes (q_s, Z_s, α_s) of n (taking into account guards and resets as made precise in the function `maxedge`);
2. if n is *tentative* with respect to (q', Z', α'), then α is equal to α'.

The result of propagation is analogous to the inequalities seen in the static guard analysis [2], however now applied to the zone graph, on-the-fly. Hence, the bounds associated to each

node (q, Z, α) never exceed those that are computed by the static guard analysis.

A delicate point about this procedure is handling of tentative nodes. When a node n is marked *tentative*, we have $\alpha = \alpha'$. However the value of α' may be updated when the tree is further explored. Thus each time we update the bounds function of a node, it is not only propagated upward in the tree but also to the nodes that are tentative with respect to n'.

This algorithm terminates as the bound functions in each node never decrease and are bounded. From the invariants above, we get that in every node, α is a solution to the equations in [2] applied on $ZG(\mathcal{A})$.

It could seem that the algorithm will be forced to do a high number of propagations of bounds. The experiments reported in Section 5 show that the present very simple approach to bound propagation is good enough. Since we propagate the bounds as soon as they are modified, most of the time, the value of α does not change in line 30 of function `propagate`. In general, bounds are only propagated on very short distances in the tree, mostly along one single edge. For this reason we do not concentrate on optimizing the function `propagate`. In the implementation we use the presented function augmented with a minor "optimization" that avoids calculating maximum over all successors in line 30 when it is not needed.

Resolution phase

Finally, as the bounds may have changed since n has been marked tentative, the function `resolve` checks for the consistency of *tentative* nodes. If $Z \subseteq Closure_{\alpha'}(Z')$ is not true anymore, n needs to be explored. Hence it is viewed as a new node: the bounds are set to $-\infty$ and n is pushed on the *stack* for further consideration in the function `main`. Setting α to $-\infty$ is safe as α will be computed and propagated when n is explored. We perform also a small optimization and propagate this bound upward, thereby making some bounds decrease.

The resolution phase may provide new nodes to be explored. The algorithm terminates when this is not the case, that is when all tentative nodes remain tentative. We can then conclude that no accepting state is reachable.

▶ **Theorem 7.** *An accepting state is reachable in $ZG(\mathcal{A})$ iff the algorithm reaches a node with an accepting state and a non-empty zone.*

4.3 Handling LU approximations

Recall that $Extra^+_{LU}(Z)$ approximation used two bounds: L_x and U_x for each clock x. In our algorithm we can easily propagate LU bounds instead of just maximal bounds. We can also replace the test $Z \subseteq Closure_{\alpha'}(Z')$ by $Z \subseteq Closure_{\alpha'}(Extra^+_{L'U'}(Z'))$, where L' and U' are the bounds calculated for (q', Z') and $\alpha'_x = \max(L'_x, U'_x)$ for every clock x. As discussed in Section 3.3, this test can be done efficiently too. The proof of correctness of the resulting algorithm is only slightly more complicated.

5 Experimental results

We have implemented the algorithm from Figure 1, and have tested it on classical benchmarks. The results are presented in Table 1, along with a comparison to UPPAAL and our implementation of UPPAAL's core algorithm that uses the $Extra^+_{LU}$ extrapolation [3] and computes bounds by static analysis [2]. Since we have not considered symmetry reduction [12] in our tool, we have not used it in UPPAAL either.

Table 1 Experimental results: number of visited nodes and running time with a timeout (T.O.) of 60 seconds. Experiments done on a MacBook with 2.4GHz Intel Core Duo processor and 2GB of memory running MacOS X 10.6.7.

Model	Our algorithm		UPPAAL's algorithm		UPPAAL 4.1.3 (-n4 -C -o1)	
	nodes	s.	nodes	s.	nodes	s.
$\mathcal{A}_1$	2	0.00	10003	0.07	10003	0.07
$\mathcal{A}_2$	7	0.00	3999	0.60	2003	0.01
$\mathcal{A}_3$	3	0.00	10004	0.37	10004	0.32
CSMA/CD7	5031	0.32	5923	0.27	–	T.O.
CSMA/CD8	16588	1.36	19017	1.08	–	T.O.
CSMA/CD9	54439	6.01	60783	4.19	–	T.O.
FDDI10	459	0.02	525	0.06	12049	2.43
FDDI20	1719	0.29	2045	0.78	–	T.O.
FDDI30	3779	1.29	4565	4.50	–	T.O.
Fischer7	7737	0.42	20021	0.53	18374	0.35
Fischer8	25080	1.55	91506	2.48	85438	1.53
Fischer9	81035	5.90	420627	12.54	398685	8.95
Fischer10	–	T.O.	–	T.O.	1827009	53.44

The comparison to UPPAAL is not meaningful for the CSMA/CD and the FDDI protocols. Indeed, UPPAAL runs out of time even if we significantly increase the time allowed; switching to breadth-first search has not helped either. We suspect that this is due to the order in which UPPAAL takes the transitions in the automaton. For this reason in columns 4 and 5, we provide results from our own implementation of UPPAAL's algorithm that takes transitions in the same order as the implementation of our algorithm. Although RED also uses approximations, it is even more difficult to draw a meaningful comparison with it, since it uses symbolic state representation unlike UPPAAL or our tool. Since this paper is about approximation methods, and not tool comparison, we leave more extensive comparisons as further work.

The results show that our algorithm provides important gains. Analyzing the results more closely we could see that both the use of closure, and on-the-fly computation of bounds are important. In Fischer's protocol our algorithm visits much less nodes. In the FDDI protocol with n processes, the DBMs are rather big square matrices of order $3n+2$. Nevertheless our inclusion test based on *Closure* is significantly better in the running time. The CSMA/CD case shows that the cost of bounds propagation does not always counterbalance the gains. However the overhead is not very high either. We comment further on the results below.

The first improvement comes from the computation of the maximal bounds used for the abstraction as demonstrated by the examples $\mathcal{A}_2$ (Figure 2), Fischer and CSMA/CD that correspond to three different situations. In the $\mathcal{A}_2$ example, the transition that yields the big bound 10^4 on y in q_0 is not reachable from any (q_0, Z), hence we just get the lower bound 20 on y in (q_0, Z), and a subsequent gain in performance.

The automaton $\mathcal{A}_1$ in Figure 2 illustrates the gain on the CSMA/CD protocol. The transition from q_0 to q_1 is disabled as it must synchronize on letter a!. The static analysis algorithm [2] ignores this fact, hence it associates bound 10^4 to y in q_0. Since our algorithm computes the bounds on-the-fly, y is associated the bound 10 in every node (q_0, Z). We observe that UPPAAL's algorithm visits 10003 nodes on $ZG(\mathcal{A}_1)$ whereas our algorithm only visits 2 nodes. The same situation occurs in the CSMA/CD example. However despite the improvement in the number of nodes (roughly 10%) the cost of computing the bounds impacts the running time negatively.

The gains that we observe in the analysis of the Fischer's protocol are explained by the automaton $\mathcal{A}_3$ in Figure 2. $\mathcal{A}_3$ has a bounded integer variable n that is initialized to 0.

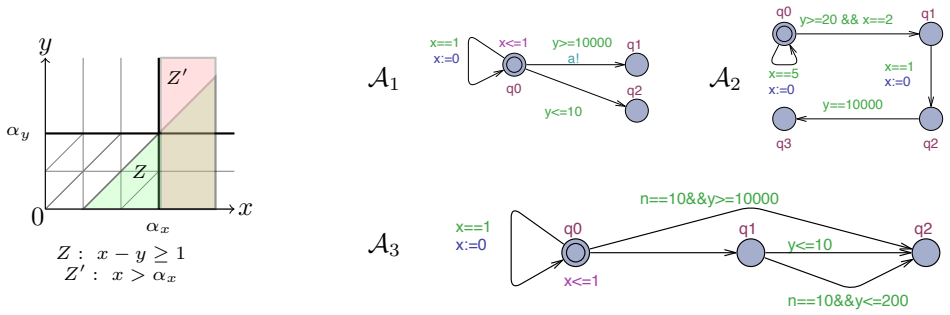


Figure 2 Examples explaining gains obtained with the algorithm.

Hence, the transitions from q_0 to q_2, and from q_1 to q_2, that check if n is equal to 10 are disabled. This is ignored by the static analysis algorithm that associates the bound 10^4 to clock y in q_0. Our algorithm however associates the bound 10 to y in every node (q_0, Z). We observe that UPPAAL's algorithm visits 10004 nodes whereas our algorithm only visits 3 nodes. A similar situation occurs in the Fischer's protocol. We include the last row to underline that our implementation is not as mature as UPPAAL. We strongly think that UPPAAL could benefit from methods presented here.

The second kind of improvement comes from the $Closure_\alpha$ abstraction that particularly improves the analysis of the Fischer's and the FDDI protocols. The situation observed on the FDDI protocol is explained in Figure 2. For the zone Z in the figure, by definition $Extra^+_{LU}(Z) = Z$, and in consequence $Z' \not\subseteq Z$. However, $Z' \subseteq Closure_\alpha(Z)$. On FDDI and Fischer's protocols, our algorithm performs better due to the non-convex approximation.

6 Conclusions

We have proposed a new algorithm for checking reachability properties of timed automata. The algorithm has two sources of improvement that are quite independent: the use of the $Closure_\alpha$ operator, and the computation of bound functions on-the-fly.

Apart from immediate gains presented in Table 1, we think that our approach opens some new perspectives on analysis of timed systems. We show that the use of non-convex approximations can be efficient. We have used very simple approximations, but it may be well the case that there are more sophisticated approximations to be discovered. The structure of our algorithm permits to calculate bounding constants on the fly. One should note that standard benchmarks are very well understood and very well modeled. In particular they have no "superfluous" constraints or clocks. However in not-so-clean models coming from systems in practice one can expect the on-the-fly approach to be even more beneficial.

There are numerous directions for further research. One of them is to find other approximation operators. Methods for constraint propagation also deserve a closer look. We believe that our approximations methods are compatible with partial order reductions [12, 14]. We hope that the two techniques can benefit from each other.

References

1 R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

2 G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen. Static guard analysis in timed automata verification. In *TACAS*, volume 2619 of *LNCS*, pages 254–270. Springer, 2003.

3 G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelanek. Lower and upper bounds in zone-based abstractions of timed automata. *Int. Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.

4 G. Behrmann, A. David, K. G Larsen, J. Haakansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *QEST'06*, pages 125–126, 2006.

5 J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. *Lectures on Concurrency and Petri Nets*, pages 87–124, 2004.

6 B. Bérard, B. Bouyer, and A. Petit. Analysing the PGM protocol with UPPAAL. *Int. Journal of Production Research*, 42(14):2773–2791, 2004.

7 P. Bouyer. Forward analysis of updatable timed automata. *Form. Methods in Syst. Des.*, 24(3):281–320, 2004.

8 C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Form. Methods Syst. Des.*, 1(4):385–415, 1992.

9 C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS'98*, volume 1384 of *LNCS*, pages 313–329. Springer, 1998.

10 D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *AVMFSS*, volume 407 of *LNCS*, pages 197–212. Springer, 1989.

11 K. Havelund, A. Skou, K. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *RTSS*, pages 2–13, 1997.

12 M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. Vaandrager. Adding symmetry reduction to UPPAAL. In *Int. Workshop on Formal Modeling and Analysis of Timed Systems*, volume 2791 of *LNCS*, pages 46–59. Springer, 2004.

13 F. Herbreteau, D. Kini, B. Srivathsan, and I. Walukiewicz. Using non-convex approximations for efficient analysis of timed automata. http://hal.archives-ouvertes.fr/inria-00559902/en/, 2011. Extended version with proofs.

14 J. Malinowski and P. Niebert. SAT based bounded model checking with partial order semantics for timed automata. In *TACAS*, volume 6015 of *LNCS*, pages 405–419, 2010.

15 G. Morbé, F. Pigorsch, and C. Scholl. Fully symbolic model checking for timed automata. In *CAV'11*, volume 6806 of *LNCS*, pages 616–632. Springer, 2011.

16 Farn Wang. Efficient verification of timed automata with BDD-like data structures. *Int. J. on Software Tools for Technology Transfer*, 6:77–97, 2004.

Shrinking Timed Automata

Ocan Sankur, Patricia Bouyer, and Nicolas Markey

LSV, CNRS and ENS Cachan, France
{sankur,bouyer,markey}@lsv.ens-cachan.fr

Abstract

We define and study a new approach to the implementability of timed automata, where the semantics is perturbed by imprecisions and finite frequency of the hardware. In order to circumvent these effects, we introduce *parametric shrinking* of clock constraints, which corresponds to tightening these. We propose symbolic procedures to decide the existence of (and then compute) parameters under which the shrunk version of a given timed automaton is non-blocking and can time-abstract simulate the exact semantics. We then define an implementation semantics for timed automata with a digital clock and positive reaction times, and show that for shrinkable timed automata, non-blockingness and time-abstract simulation are preserved in implementation.

1998 ACM Subject Classification D.2.4 Software/Program Verification; F.1.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases timed automata, implementability, robustness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.90

1 Introduction

Timed automata [3] are a well-established model in real-time system design. They offer an automata-theoretic framework to design, verify and synthesize systems with timing constraints. The theory behind timed automata has been extensively studied and mature model-checking tools are available. However, this model makes unrealistic assumptions on the system, such as the perfect continuity of clocks and instantaneous reaction times, which are not preserved in implementation even in digital hardware with arbitrary finite precisions. Often, the *synchrony hypothesis* allows one to ignore this issue and greatly simplifies the design phase [7]. However, the synchrony hypothesis still needs to be formally validated once the design phase is over. In fact, perturbations on clocks, either imprecisions or clock drifts, however small they may be, may yield extra qualitative behaviours in some timed systems [22, 14]; positive reaction times can also disable desired behaviours [11, 1].

This raises the question of *implementability*, *i.e.*, whether the model can be implemented on physical machines, preserving (the properties of) its exact semantics. In order to model the behaviour of *implementations* of timed automata, and validate the synchrony hypothesis, De Wulf *et al.* introduced the *program semantics* for timed automata, which defines the behaviour of a timed automaton on a simple micro-processor with a digital clock [15]. This semantics is a bit jagged, and the *enlarged semantics* has been proposed as a convenient over-approximation: it models imprecisions by relaxing all guards, turning clock constraints of the form $x \in [a, b]$ into $x \in [a - \Delta, b + \Delta]$ for some positive parameter Δ. *Robust model checking*, which consists in deciding the existence of a value for Δ under which a property is satisfied in the enlarged semantics, has been proven decidable for safety properties [14], and for richer linear-time properties [9, 10]. In this framework, the implementation (the program

Partly supported by project ImpRo (ANR-2010-BLAN-0317) of the French National Agency for Research.

semantics) always has more behaviours than the abstract model (the exact semantics), due to the relaxation of the timing constraints, and robust model-checking only ensures correctness for properties preserved by timed simulation.

We adopt the following approach: instead of checking properties on the enlarged semantics of a given timed automaton $\mathcal{A}$, we look for a new timed automaton $\mathcal{B}$ whose semantics would *correspond* to the (exact) semantics of $\mathcal{A}$. To circumvent the effect of the imprecisions, our idea is to construct $\mathcal{B}$ by *shrinking* the guards of $\mathcal{A}$, which is the opposite to enlargement, so that all behaviours of $\mathcal{B}$ under enlargement are included in those of $\mathcal{A}$. The timed automaton $\mathcal{B}$ constructed in this manner preserves in particular all universal properties (like linear-time properties) proven (say, by model-checking) for $\mathcal{A}$. This also means that all timing requirements satisfied by $\mathcal{A}$, such as critical deadlines, are strictly respected by $\mathcal{B}$. However, such a transformation may remove too many behaviours and even introduce deadlocks in $\mathcal{B}$. The preservation of the desired behaviours in $\mathcal{B}$ is the problem we are interested in.

A timed automaton $\mathcal{A}$ is said *shrinkable* if it can be shrunk into a timed automaton that is non-blocking, and/or can time-abstract-simulate $\mathcal{A}$. We do not restrict to one single shrinking parameter, but to one parameter per atomic clock constraint in the automaton. We give algorithms to decide the existence of these shrinking parameters, and compute the least parameters when they exist. We show that shrinkability w.r.t. non-blockingness can be checked in PSPACE, shrinkability w.r.t. simulation in EXPTIME, and shrinkability (w.r.t. both requirements) in EXPTIME, by symbolic procedures manipulating difference bound matrices.

As a second result, we define an implementation semantics of timed automata executed by a digital system with a digital clock. Our semantics is similar to the program semantics of [15] but it is valid under slightly different assumptions. We study the relations between the exact semantics and the implementation semantics, and prove additional properties besides the one given in [15]. We show that when a timed automaton $\mathcal{A}$ is shrinkable, say to a timed automaton $\mathcal{B}$, then the implementation semantics of $\mathcal{B}$ is non-blocking and time-abstract-simulates the exact semantics of $\mathcal{A}$. Thus, our framework allows not only to obtain an implementation that contains no more behaviour than the abstract model but also to ensure non-blockingness and time-abstract similarity. This provides a precise motivation for the shrinkability problem: shrinkability is a sufficient condition for the correctness of the implementation semantics.

Finally, notice that shrinkability is also an interesting property by itself, since it asks whether the given automaton is vulnerable to infinitesimal shrinkings (which can be due to imprecisions). Zeno behaviours and other different convergence phenomena [11] are also naturally excluded in shrunk systems (see Section 3.2).

2 Preliminaries

A *timed transition system (TTS)* is a tuple $(S, s_0, \Sigma, \to)$, where S is the set of *states*, $s_0 \in S$ the initial state, Σ a finite alphabet, and $\to \subseteq S \times (\Sigma \times \mathbb{R}_{\geq 0}) \times S$ the *transitions*. Transitions are labelled by $\sigma(T)$, with $T \in \mathbb{R}_{\geq 0}$ the *timestamp* of *action* $\sigma \in \Sigma$. In all TTSs we consider, the timestamps of consecutive actions are assumed to be nondecreasing. A TTS $(S, s_0, \Sigma, \to)$ is *non-blocking* if for any transition $s_1 \xrightarrow{\sigma(T)} s_2$, there exist $\sigma' \in \Sigma$, $T' \geq T$ and $s_3 \in S$ such that $s_2 \xrightarrow{\sigma'(T')} s_3$. Notice that, in this definition, we do not require s_1 to be reachable from s_0.

▶ **Definition 1.** Let $\mathcal{S} = (S, s_0, \Sigma, \to)$ be a TTS. A relation $R \subseteq S \times S$ is a *timed* (resp.

time-abstract) *simulation* if for all $(s_1, s_2) \in R$, if $s_1 \xrightarrow{\sigma(T)} s_1'$ for some $(\sigma, T) \in \Sigma \times \mathbb{R}_{\geq 0}$, then $s_2 \xrightarrow{\sigma(T)} s_2'$ (resp. $s_2 \xrightarrow{\sigma(T')} s_2'$ for some $T' \in \mathbb{R}_{\geq 0}$) for some s_2' with $(s_1', s_2') \in R$. A state s_2 *timed-simulates* (resp. *time-abstract-simulates*) a state s_1 if there exists a timed (resp. time-abstract) simulation R such that $(s_1, s_2) \in R$. In that case, we write $s_1 \sqsubseteq s_2$ (resp. $s_1 \sqsubseteq_{\text{t.a.}} s_2$).

Given two TTSs $\mathcal{S}$ and $\mathcal{T}$, we write $\mathcal{S} \sqsubseteq \mathcal{T}$ if the initial state of $\mathcal{T}$ timed-simulates that of $\mathcal{S}$ in their disjoint union. We write $\mathcal{S} \sqsubseteq_{\text{t.a.}} \mathcal{T}$ in case of an time-abstract simulation. For any state s of $\mathcal{S}$, we write $\text{ta-sim}_{\mathcal{T}}(s)$ for the set of states of $\mathcal{T}$ that time-abstract simulate s. This set is called *the (time-abstract) simulator set of s in $\mathcal{T}$*.

Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}_{\geq 0}^{\mathcal{C}}$. For a subset $R \subseteq \mathcal{C}$, a real number $\alpha \in \mathbb{R}_{\geq 0}$ and a valuation v, we write $v[R \leftarrow \alpha]$ for the valuation defined by $v[R \leftarrow \alpha](x) = v(x)$ for $x \in \mathcal{C} \setminus R$ and $v[R \leftarrow \alpha](x) = \alpha$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way.

Let $\mathbb{Q}_\infty = \mathbb{Q} \cup \{-\infty, \infty\}$. An *atomic clock constraint* is a formula of the form $k \leq x \leq l$ or $k \leq x - y \leq l$ where $x, y \in \mathcal{C}$ and $k, l \in \mathbb{Q}_\infty$. A *guard* is a conjunction of atomic clock constraints. We denote by $\Phi_{\mathcal{C}}$ the set of guards on the clock set $\mathcal{C}$. We define the *enlargement* of atomic clock constraints by $\delta \in \mathbb{Q}$ as follows: for $x, y \in \mathcal{C}$ and $k, l \in \mathbb{Q}_{>0}$, we let

$$\langle k \leq x - y \rangle_\delta = k - \delta \leq x - y, \qquad \langle x - y \leq l \rangle_\delta = x - y \leq l + \delta.$$

(and similarly for $\langle k \leq x \rangle_\delta$ and $\langle x \leq l \rangle_\delta$). The enlargement of a guard g, denoted by $\langle g \rangle_\delta$, is obtained by enlarging all its atomic clock constraints. Notice that δ can be negative here; this operation is then called *shrinking*. A valuation v *satisfies* a guard g, denoted $v \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced by $v(x)$. We denote by $[\![g]\!]$ the set of valuations that satisfy g. We will write $v \models_\delta g$ to mean $v \models \langle g \rangle_\delta$.

▶ **Definition 2.** A timed automaton $\mathcal{A}$ is a tuple $(\mathcal{L}, l_0, \mathcal{C}, \Sigma, E)$, with finite sets $\mathcal{L}$ of *locations*, $\mathcal{C}$ of *clocks*, Σ of *labels*, $E \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times \Sigma \times 2^{\mathcal{C}} \times \mathcal{L}$ of edges, with $l_0 \in \mathcal{L}$ the *initial location*. An edge $e = (l, g, \sigma, R, l')$ is also written as $l \xrightarrow{g,\sigma,R} l'$. Guard g is called the *guard* of e.

For any timed automaton $\mathcal{A}$, let $(g_i)_{i \in I}$ denote the vector of all atomic clock constraints used in its guards. Given a vector of rational numbers $\overline{\delta} = (\delta_i)_{i \in I}$, we define $\mathcal{A}_{\overline{\delta}}$ as the timed automaton obtained from $\mathcal{A}$ by replacing g_i with $\langle g_i \rangle_{\delta_i}$. For any $\Delta \in \mathbb{Q}$, $\mathcal{A}_\Delta$ will denote the timed automaton where all guards are enlarged by Δ.

▶ **Definition 3.** The semantics of a timed automaton $\mathcal{A} = (\mathcal{L}, l_0, \mathcal{C}, \Sigma, E)$ is a TTS over alphabet Σ, denoted $[\![\mathcal{A}]\!]$, whose state space is $\mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}} \times \mathbb{R}_{\geq 0}$. The initial state is $(l_0, \overline{0}, 0)$, where $\overline{0}$ denotes the valuation where all clocks have value 0. There is a transition $(l, v, T) \xrightarrow{\sigma(T+\tau)} (l', v', T + \tau)$, for any edge $l \xrightarrow{g,\sigma,R} l'$ and $\tau \geq 0$, such that $v + \tau \models g$ and $v' = (v + \tau)[R \leftarrow 0]$.

We assume familiarity with the usual notions of *region equivalence* and *region automaton*, and refer to [3] for definitions. The important property used here is that time-abstract-simulating a timed automaton is equivalent to (time-abstract-)simulating its region automaton.

3 Shrinkability

3.1 Robustness and Shrinking

Robust model-checking, that is, the analysis of timed automata under clock imprecisions have been studied in [22, 14, 9, 10, 21, 23]. It is shown in [15] how this framework allows one to *validate the synchrony hypothesis*, that is, prove that the semantics is preserved in a physical implementation with imprecisions. See [22, 14] for examples of timed automata that are *not robust*: their behaviours change in presence of the slightest positive guard enlargement. In a recent work [8], we defined transformations that provide, for any given timed automaton $\mathcal{A}$, a timed automaton $\mathcal{A}'$ such that $\mathcal{A}$ and $\mathcal{A}'_\Delta$ are ϵ-bisimilar, that is, there is a timed bisimulation in which the differences in delays are bounded by ϵ at each step. The advantage of that approach is that it works for all timed automata, and that we obtain an ϵ-bisimilar enlarged timed automaton, for any desired $\epsilon > 0$. However, the timed behaviour of the resulting automaton may not be included in the abstract model; it is only preserved approximately. Also, the size of $\mathcal{A}'$ is exponential, and we do not make the link with an implementation semantics.

We now define *shrinking* of timed automata and show how it provides an alternative way to construct robust systems. Our method provides a construction of the same size as the initial automaton, and whose timed behaviour is always included in the abstract model. Our algorithms then allow to decide whether further properties, such as non-blockingness and time-abstract similarity, can be satisfied. In order to circumvent the effect of the imprecisions, we propose to *shrink* any guard of the form "$x \in [a, b]$" into "$x \in [a+\delta, b-\delta]$" for some $\delta > 0$, so that under a small enlargement parameter $\Delta > 0$, we have $[a + \delta - \Delta, b - \delta + \Delta] \subseteq [a, b]$; in other terms, the satisfaction of the enlarged guard implies the satisfaction of the original guard. Formally, we will consider the *shrunk timed automaton* $\mathcal{A}_{-\mathbf{k}\delta}$ where $\mathbf{k} = (k_i)_{i\in I} \in \mathbb{N}^I_{>0}$ and $\delta > 0$. Clearly, if $\Delta < \max_{i\in I}(k_i) \cdot \delta$, $\mathcal{A}_{-\mathbf{k}\delta+\Delta}$ does not contain more behaviours than $\mathcal{A}$; in fact $[\![\mathcal{A}_{-\mathbf{k}\delta+\Delta}]\!] \sqsubseteq [\![\mathcal{A}]\!]$.

Shrinking is a natural idea when one is interested in the preservation of strict timing constraints, such as critical deadlines. However, shrinking may remove too many behaviours and the resulting automaton may even become blocking. We are interested in deciding the existence of shrinking parameters $\mathbf{k}$ and δ, and in their computation, for which the shrunk timed automaton is non-blocking and/or is able to time-abstract simulate the original automaton. We will see in Section 6 that when the shrunk automaton satisfies these properties, these are preserved in a concrete implementation semantics.

▶ **Definition 4.** A timed automaton $\mathcal{A}$ is *shrinkable* if there exists $\mathbf{k} \in \mathbb{N}^I_{>0}$ and $\delta_0 \in \mathbb{Q}_{>0}$ such that for all $0 \leq \delta \leq \delta_0$,

- $[\![\mathcal{A}_{-\mathbf{k}\delta}]\!]$ is non-blocking,
- $[\![\mathcal{A}]\!] \sqsubseteq_{\text{t.a.}} [\![\mathcal{A}_{-\mathbf{k}\delta}]\!]$, with the following additional technical requirement: for each region (l, r) of $\mathcal{R}(\mathcal{A})$, there is a guard g and a vector $\mathbf{h}$ of non-negative integers such that for all $0 \leq \delta \leq \delta_0$, the simulator set of (l, r) in $[\![\mathcal{A}_{-\mathbf{k}\delta}]\!]$ equals $[\![\langle g\rangle_{-\mathbf{h}\delta}]\!]$.

We say that $\mathcal{A}$ is *shrinkable w.r.t. non-blockingness* (resp. *w.r.t. simulation*) if only the first (resp. second) condition holds. The above $\mathbf{k}$ and δ_0 are shrinking parameters for $\mathcal{A}$.

We define shrinkability "for all $0 \leq \delta \leq \delta_0$", so if an automaton is shrinkable, we require it to remain correct when imprecisions are reduced, that is when δ is chosen smaller. In fact, the shrunk automaton can be seen as an approximation of the initial automaton, and we would like to be able to obtain arbitrarily close correct approximations by only adjusting δ.

This requirement is also related to the property called "faster-is-better" [2, 15]. Notice also that when a timed automaton is shrinkable w.r.t. simulation, then we require that for all small enough δ, each simulator set can be expressed as shrinkings $\langle g \rangle_{-\mathbf{h}\delta}$ where $\mathbf{h}$ is the same for all δ (that is, parameters $\mathbf{h}$ are *uniform*). Then, when we adjust the parameter δ, the expressions of the simulator sets do not change. This is desirable for instance when one needs to use these constraints in the system.

3.2 Shrinking as a Remedy to Unrealistic Behaviour

Shrinkability also excludes *unrealistic* timing constraints, such as Zeno behaviours. In fact, for any timed automaton $\mathcal{A}$, consider the automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by adding a new clock u, the constraint $u \geq 0$ and the reset $u := 0$ at every edge. Clearly, $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic. If automaton $\mathcal{A}'$ is shrinkable, then $\mathcal{A}$ does not need Zeno strategies to satisfy the properties proven for the exact semantics and preserved by time-abstract similarity (in fact, each $u \geq 0$ is shrunk to some $u \geq \delta_i$ with $\delta_i > 0$).

But unrealistic timing constraints are not limited to Zeno behaviours. The automaton in Fig. 1 provides an example of a timed automaton which is non-blocking for $\delta_1 = \delta_2 = \delta_3 = 0$, and lets the time diverge but it becomes blocking whenever $\delta_2 > 0$ or $\delta_3 > 0$, so it is not shrinkable. A similar example was provided in [11] but with equality constraints, so it is trivially not shrinkable. In section 5, we give an example of a shrinkable timed automaton (Fig. 3).

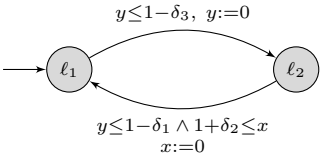


Figure 1 A shrunk timed automaton that is blocking whenever $\delta_2 > 0$ or $\delta_3 > 0$.

3.3 Decidability of Shrinkability

Our main result is the decidability of shrinkability:

▶ **Theorem 5.** *Shrinkability w.r.t. non-blockingness can be decided in* PSPACE*, and in* NP *if the number of outgoing transitions from each location is bounded. Shrinkability w.r.t. simulation is decidable in* EXPTIME*. Finally, shrinkability is decidable in* EXPTIME*.*

Moreover, we will show that when a given timed automaton is shrinkable, the least shrinking parameters can be computed (see Section 5 for details). In the rest, we present the proof of this result. We begin by defining parametric *difference-bound matrices* (DBMs) and give tools for solving fixpoint equations on DBMs through max-plus equations. We then explain how this can be used to decide shrinkability. In Section 6, we present a concrete implementation semantics and prove that non-blockingness and simulation are preserved in this semantics for all shrinkable timed automata.

4 Some algebraic tools

4.1 Parameterized Difference Bound Matrices

Difference bound matrices are data structures used to represent sets of clock valuations in timed automata analysis [17]. Write $\mathcal{C} = \{1, \ldots, C\}$, and add an artificial clock of index 0, that has constant value 0. We let $\mathcal{C}_0 = \mathcal{C} \cup \{0\}$. A *difference bound matrix* (DBM) over $\mathcal{C}_0$ is

an element of $\mathcal{M}_{C+1}(\mathbb{Q}_\infty)$[1]. Each $M \in \mathcal{M}_{C+1}(\mathbb{Q}_\infty)$ defines a *zone*, that is, a convex subset of $\mathbb{R}^{\mathcal{C}}_{\geq 0}$ defined by $[\![M]\!] = \{v \in \mathbb{R}^{\mathcal{C}}_{\geq 0} \mid \forall x, y \in \mathcal{C}_0, -M_{y,x} \leq v(x) - v(y) \leq M_{x,y}\}$. Clearly, each DBM can be equivalently described by a guard, and conversely. A DBM M is *normalized* when for all $x, y, z \in \mathcal{C}_0$, it holds $M_{x,y} \leq M_{x,z} + M_{z,y}$. Any non-empty DBM can be made normalized in polynomial time, by interpreting it as an adjacency matrix of a weighted graph and computing all shortest paths between any two clocks.

We define several *elementary operations* on DBMs. Given a DBM M, we let $\mathrm{Pre}_{\mathsf{time}}(M)$ be the normalized DBM that describes the time predecessors of $[\![M]\!]$, *i.e.*, $[\![\mathrm{Pre}_{\mathsf{time}}(M)]\!] = \{v \in \mathbb{R}^{C}_{\geq 0} \mid \exists t \in \mathbb{R}_{\geq 0} \text{ s.t. } v + t \in [\![M]\!]\}$. Given $R \subseteq \mathcal{C}$, we let $\mathrm{Unreset}_R(M)$ be the normalized DBM that defines $\{v \in \mathbb{R}^{C}_{\geq 0} \mid v[R \leftarrow 0] \in [\![M]\!]\}$. For two DBMs M and N, we write $M \cap N$ for the normalized DBM describing $[\![M]\!] \cap [\![N]\!]$. A function $f \colon \mathcal{M}_{C+1}(\mathbb{Q}_\infty)^n \to \mathcal{M}_{C+1}(\mathbb{Q}_\infty)$ (for some $n > 0$), is said *elementary* if it combines its arguments using elementary operations. Efficient algorithms exist for computing these operations on DBMs [6, 12].

We extend standard DBMs in order to manipulate sets of states in shrunk timed automata. We fix a tuple of parameters $\mathbf{k} = (k_i)_{i \in I}$, which will take nonnegative integer values. The *max-plus polynomials* over $\mathbf{k}$, denoted by $\mathcal{G}(\mathbf{k})$, are generated by the grammar $\phi ::= l \in \mathbb{N} \mid k_i,\ i \in I \mid \phi + \phi \mid \max(\phi, \phi)$. For any max-plus polynomial ϕ and valuation $\nu \colon \mathbf{k} \to \mathbb{N}$, we denote by $\phi[\nu]$ the value of formula ϕ replacing each parameter k by $\nu(k)$. A (resp. *positive*, *parameterized*) *shrinking matrix* is an element of $\mathcal{M}_{C+1}(\mathbb{N})$ (resp. $\mathcal{M}_{C+1}(\mathbb{N}_{>0})$, $\mathcal{M}_{C+1}(\mathcal{G}(\mathbf{k}))$). If P is a parameterized shrinking matrix (PSM) and ν is a valuation, the shrinking matrix $P[\nu]$ is defined in a natural way. Note that our definition of PSM is different from parametric DBMs considered for instance in [20], since we use max-plus polynomials instead of linear expressions and only consider natural number valuations. In what follows, we manipulate parameterized DBMs, also called *shrunk DBMs*, of the form $M - \delta \cdot P$, where δ is a fresh parameter, and P is a PSM. If M is a DBM for guard g, the shrunk guard $\langle g \rangle_{-\delta}$ will be represented by the shrunk DBM $M - \mathbf{1} \cdot \delta$, where matrix $\mathbf{1}$ has 0's on the diagonal and 1's everywhere else.

Shrunk DBMs will be used as a data structure for manipulating the state space of shrunk timed automata. The following lemma explains how elementary operations can be computed on shrunk DBMs. In particular, it shows how shrinking parameters (*i.e.*, the PSM) can be propagated in a backward analysis while staying in the max-plus theory.

▶ **Lemma 6.** *Let $M, M_1, \ldots, M_n$ be non-empty normalized DBMs and $f \colon \mathcal{M}_{C+1}(\mathbb{Q}_\infty)^n \to \mathcal{M}_{C+1}(\mathbb{Q}_\infty)$ be an elementary function with $M = f(M_1, \ldots, M_n)$. Let $P_1, \ldots, P_n$ be matrices in $\mathcal{M}_{C+1}(\mathcal{G}(\mathbf{k}))$. Then, we can compute $P' \in \mathcal{M}_{C+1}(\mathcal{G}(\mathbf{k}))$ s.t. for all $\nu \colon \mathbf{k} \to \mathbb{N}$, there exists a (computable) $\delta_0 > 0$ s.t. $M - \delta P'[\nu] = f(M_1 - \delta P_1[\nu], \ldots, M_n - \delta P_n[\nu])$ for all $0 \leq \delta < \delta_0$. All these computations can be achieved in polynomial time, and in particular P' has size polynomial in the size of $P_1, \ldots, P_n$ and f. If the above property holds, we write $M - \delta P' = f(M_1 - \delta P_1, \ldots, M_n - \delta P_n)$.*

For solving the shrinkability problems, we will use fixpoint equations on shrunk DBMs (see Section 5). As a prerequisite, we therefore first investigate max-plus equations, and then give a general theorem for solving those equations.

[1] $\mathcal{M}_n(X)$ is the set of $n \times n$ matrices with coefficients in X, where all diagonal coefficients are 0.

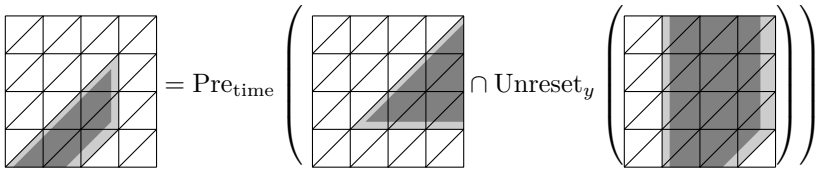


Figure 2 Consider an edge $\ell \xrightarrow{g,\sigma,R} \ell'$ in a timed automaton where $g = 1 \leq y \wedge 0 \leq x - y$ and $R = \{y\}$. For any pair of zones X, Y, the equation $Y = \mathrm{Pre}_{\mathsf{time}}([\![g]\!] \cap \mathrm{Unreset}_y(X))$ expresses the fact that X can be reached in one step starting from Y. Consider $Y = [\![0 \leq x, y \leq 3 \wedge 0 \leq x - y \leq 2]\!]$, and $X = [\![1 \leq x \leq 4 \wedge x - y \leq 3]\!]$. In the figure, the union of dark gray and light gray areas illustrate this equation while the dark gray areas illustrate the equation between *shrunk* zones. Let us assume that we shrink g to $g' = 1 + k_1\delta \leq y \wedge k_2\delta \leq x - y$ and X to $X' = [\![1 + k_3\delta \leq x \leq 4 - k_4\delta \wedge x - y \leq 3 - k_5\delta]\!]$, for positive integers k_i and $\delta > 0$ small enough so that these sets are non-empty. Then, by Lemma 6, we can compute the shrinking parameters for Y, so that the shrunk zone Y' satisfies the new equation $Y' = \mathrm{Pre}_{\mathsf{time}}([\![g']\!] \cap \mathrm{Unreset}_y(X'))$. We get:

$$\begin{aligned}
&\mathrm{Unreset}_y(X') && = [\![1 + k_3\delta \leq x \leq 3 - k_5\delta]\!],\\
&[\![g']\!] \cap \mathrm{Unreset}_y(X') && = [\![1 + \max(k_1 + k_2, k_3)\delta \leq x \leq 3 - k_5\delta\\
& && \quad \wedge 1 + k_1\delta \leq y \leq 3 - (k_2 + k_5)\delta\\
& && \quad \wedge k_2\delta \leq x - y \leq 2 - (k_1 + k_5)\delta]\!],\\
&Y' = \mathrm{Pre}_{\mathsf{time}}([\![g']\!] \cap \mathrm{Unreset}_y(X')) && = [\![k_2\delta \leq x \leq 3 - k_5\delta \wedge 0 \leq y \leq 3 - (k_2 + k_4)\delta\\
& && \quad \wedge k_2\delta \leq x - y \leq 2 - (k_1 + k_5)\delta]\!].
\end{aligned}$$

This equality holds for all $0 \leq \delta < \min(\frac{1}{k_4 - k_5}, \frac{2}{\max(k_1+k_2,k_3)+k_5}, \frac{3}{k_2+k_5}, \frac{2}{k_1+k_2+k_5}, \frac{2}{k_1-k_2+k_5})$, where a term is $+\infty$ if the denominator is zero.

4.2 Max-plus equations

In PSMs, formal expressions using maximization and sum are manipulated. The set $\mathbb{R}_{\geq 0}$ endowed with these operations is called the *max-plus algebra*. There is a well-established theory on solving equations in this algebra, with applications to discrete-event systems [5].

Let $k_1, \ldots, k_n, k_{n+1}, \ldots, k_{n+n'}$ be parameters, and $\phi_1, \ldots, \phi_n$ be max-plus polynomials. We will be interested in computing solutions of fixpoint equations of the following form:

$$k_i = \phi_i(k_1, \ldots, k_n, k_{n+1}, \ldots, k_{n+n'}), \quad \forall 1 \leq i \leq n. \tag{E}$$

Notice that variables $k_{n+1}, \ldots, k_{n+n'}$ only appear at the right hand side of the equation. Equation (E) defines a *non-linear* equation (polynomials ϕ_i have arbitrary degrees). Although Tarski's Theorem [24] guarantees the existence of fixpoint solutions in $\mathbb{N} \cup \{\infty\}$, we are interested in *finite* solutions, *i.e.*, solutions in $\mathbb{N}$ which is not a complete lattice.

▶ **Theorem 7.** *For any equation of the form* (E)*, the existence of a solution in* $\mathbb{N}$ *is decidable in polynomial time in the size of the equation. Moreover, assume there is a solution in* $\mathbb{N}$ *in which* $k_{n+1}, \ldots, k_{n+n'}$ *take positive values; then given any fixed positive values* $v_{n+1}, \ldots, v_{n+n'}$*, Equation* (E) *with the additional constraints* $k_{n+i} = v_{n+i}$ *for all* $1 \leq i \leq n'$ *has a least solution, computable in polynomial time.*

The second point of Theorem 7 states that the existence of solutions with positive values for the unconstrained variables does not depend on their exact values. These results rely on an analysis of max-plus graphs, that we associate to max-plus equations.

4.3 Equations on shrunk DBMs

We now apply the previous results to solving equations on shrunk DBMs. We consider fixpoint equations on DBMs of the form:

$$M_i = f_i(M_1, \dots, M_n, M_{n+1}, \dots, M_{n+n'}), \quad \forall 1 \leq i \leq n, \tag{1}$$

where $M_1, \dots, M_{n+n'}$ are unknown normalized DBMs ($M_{n+1}, \dots, M_{n+n'}$ are unconstrained) and f_i's are elementary functions. We are interested in *shrunk solutions* defined as follows.

▶ **Definition 8.** Fix a solution $(M_i)_i$ of (1). A *shrunk solution* of (1) w.r.t. $(M_i)_i$ is a triple $\big((M_i)_i, (Q_i)_i, \delta_0\big)$, where $\delta_0 > 0$ and Q_i's are shrinking matrices such that for all $0 \leq \delta \leq \delta_0$, $(M_i - \delta Q_i)_i$ is a solution of (1). A shrunk solution is called the *greatest shrunk solution* if $(Q_i)_i$ are the least shrinking matrices which define a shrunk solution w.r.t. $(M_i)_i$.

Assume that (1) has a solution and fix one, say $(M_i)_i$. From Lemma 6, there exist matrices $(\phi_i)_{1\leq i\leq n}$ of max-plus polynomials s.t. for all shrinking matrices $(Q_i)_i$, there exists $\delta_0 > 0$ such that $M_j - \phi_j((Q_i)_i) \cdot \delta = f_j((M_i - Q_i \cdot \delta)_i)$ for all $0 \leq \delta \leq \delta_0$ and all $1 \leq j \leq n$. This suggests that we study the following fixpoint equation on PSMs P_i's, where each coefficient is a fresh parameter and polynomials ϕ_i's are those from Lemma 6 for these P_i's:

$$P_i = \phi_i(P_1, \dots, P_n, P_{n+1}, \dots, P_{n+n'}), \quad \forall 1 \leq i \leq n. \tag{2}$$

This is a max-plus equation (like (E)), whose size is polynomial in the size of (1). The following lemma links the shrunk solutions of (1) with the solutions of (2).

▶ **Lemma 9.** *Fix any solution $(M_i)_i$ of (1) and consider max-plus polynomial matrices $(\phi_i)_{1\leq i\leq n}$ as defined above. Then,*

- *For all shrinking matrices $(Q_i)_i$, there exists $\delta_0 > 0$ s.t. $\big((M_i)_i, (Q_i)_i, \delta_0\big)$ is a shrunk solution of (1) if, and only if, $(Q_i)_i$ is a solution of (2) in $\mathbb{N}$.*
- *$\big((M_i)_i, (Q_i)_i, \delta_0\big)$ is the greatest shrunk solution of (1) iff $(Q_i)_i$ is the least solution of (2).*
- *If (2) has a solution $(Q_i)_i$ where $Q_{n+1}, \dots, Q_{n+n'}$ have positive coefficients (except for 0 on the diagonal), then for any matrices $R_{n+1}, \dots, R_{n+n'}$ in $\mathcal{M}_{C+1}(\mathbb{N}_{>0})$, (2) with the additional constraints $P_{n+i} = R_{n+i}$ for all $1 \leq i \leq n'$ has a least shrunk solution, computable in polynomial time.*

Notice that by Lemma 9, one can also decide the existence of a solution of (2), where all $Q_{n+1}, \dots, Q_{n+n'}$ are positive shrinking matrices: it suffices to add to (2) the equalities $Q_{n+i} = \mathbf{1}$, where $\mathbf{1}$ is the matrix with 1's everywhere except for 0's on the diagonal.

5 Deciding shrinkability

We now apply the results we developed in previous sections to shrinkability. We fix a non-blocking timed automaton $\mathcal{A} = (\mathcal{L}, l_0, \mathcal{C}, \Sigma, E)$. We assume that all edges of $\mathcal{A}$ have distinct labels, and identify edges with their labels. This is harmless for our purpose since we compare an automaton to its shrinking that has the same structure. For any edge with label $\sigma \in \Sigma$ and guard g_σ, let G_σ be the DBM that represents $[\![g_\sigma]\!]$, and R_σ the reset set.

5.1 Shrinkability w.r.t. simulation.

Thanks to the hypothesis on distinct edge labels, the simulator sets of regions (l, r) in $[\![\mathcal{A}]\!]$ can be expressed by the following fixpoint equation:

$$[\![M_{l,r}]\!] = \bigcap_{\sigma\in\Sigma} \bigcap_{(l,r)\xRightarrow{\sigma}(l',r')} \mathrm{Pre}_{\mathsf{time}}(\mathrm{Unreset}_{R_\sigma}([\![M_{l',r'}]\!]) \cap [\![G_\sigma]\!]), \tag{3}$$

for all $(l,r) \in \mathcal{R}(\mathcal{A})$, where $(M_{l,r})_{l,r}$ are the unknown DBMs. In the greatest solution, each $M_{l,r}$ represents the simulator set of region (l,r) in $[\![\mathcal{A}]\!]$. Consider the greatest solution $(M_{l,r})_{l,r} \cup (G_\sigma)_\sigma$, where we see G_σ's as a part of the solution. Let $(l_0, \vec{0})$ denote the initial state of $\mathcal{R}(\mathcal{A})$. Solving shrinkability means deciding whether (3) has a shrunk solution with respect to $(M_{l,r})_{l,r} \cup (G_\sigma)_\sigma$, such that $M_{l_0,\vec{0}}$ is shrunk to a zone that contains $\vec{0}$ (since $(l_0,\vec{0})$ is the initial state of any shrinking of $\mathcal{A}$). For any shrinking matrix $P_{l_0,\vec{0}}$, it can be checked in polynomial time whether $\vec{0}$ belongs to $[\![M_{l_0,\vec{0}} - \delta P_{l_0,\vec{0}}]\!]$ for sufficiently small δ. Now, if there is a shrunk solution to (3), then for any positive shrinking matrices $(K_\sigma)_\sigma$ Lemma 9 provides a (greatest) shrunk solution where the shrinking matrices for $(G_\sigma)_\sigma$ are fixed to $(K_\sigma)_\sigma$. In particular, shrinkability w.r.t. simulation does not depend on how much guards are shrunk: either all positive integer vectors $\mathbf{k}$ witness the shrinkability of $\mathcal{A}$ (into $\mathcal{A}_{-\mathbf{k}\delta}$), or $\mathcal{A}$ is not shrinkable w.r.t. simulation for any value of $\mathbf{k}$.

The simulator sets $M_{l,r}$ can be computed in exponential time ([18]), and Equation (3) has size polynomial in the size of these sets. By Lemma 9, the overall complexity is in EXPTIME.

5.2 Shrinkability w.r.t. non-blockingness.

Since automaton $\mathcal{A}$ is non-blocking, $(G_\sigma)_\sigma$ satisfies the following equation.

$$\forall \sigma \in \Sigma, \quad [\![G_\sigma]\!] \subseteq \bigcup_{\sigma' : (\sigma,\sigma') \in \Sigma_{E \circ E}} \mathrm{Unreset}_{R_\sigma}(\mathrm{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])), \tag{4}$$

where we let $\Sigma_{E\circ E} = \{(\sigma,\sigma') \mid \exists l, l', l'' \in \mathcal{L}, l \xrightarrow{\sigma} l' \xrightarrow{\sigma'} l'' \in E\}$, that is the set of pairs of labels of consecutive transitions in $\mathcal{A}$. We rewrite this equivalently as follows.

$$\forall \sigma \in \Sigma, \quad [\![G_\sigma]\!] = \bigcup_{\sigma' : (\sigma,\sigma') \in \Sigma_{E \circ E}} \mathrm{Unreset}_{R_\sigma}(\mathrm{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])) \cap [\![G_\sigma]\!], \tag{5}$$

Now, $\mathcal{A}$ is shrinkable w.r.t. non-blockingness if, and only if, this equation has a shrunk solution w.r.t. $(G_\sigma)_\sigma$. We can unfortunately not directly use our general results on shrunk solutions since our equation contains a union. We instead apply transformations to this equation in order to remove the union. We start by rewriting the above equation as follows:

$$\forall \sigma \in \Sigma, \quad [\![G_\sigma]\!] = \bigcup_{\sigma' : (\sigma,\sigma') \in \Sigma_{E \circ E}} [\![M_{\sigma,\sigma'}]\!], \qquad [\![M_{\sigma,\sigma'}]\!] = \mathrm{Unreset}_{R_\sigma}(\mathrm{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])) \cap [\![G_\sigma]\!]. \tag{6}$$

Fix a solution $(G_\sigma)_\sigma \cup (M_{\sigma,\sigma'})_{\sigma,\sigma'}$, which exists again by the non-blockingness assumption. We solve the max-plus equation corresponding to the right part of (6) by Lemma 9, but we will add to this equation some inequalities which "encode" the left part of (6). We use the following technical lemma to choose these inequalities.

▶ **Lemma 10.** *Let $C_1, \ldots, C_b$ and D be normalized DBMs s.t. $[\![D]\!] = \bigcup_{1\le i\le b} [\![C_i]\!]$ and $P_1, \ldots, P_b$ and Q shrinking matrices s.t. for some $\delta_0 > 0$, $D - \delta Q$ and $C_i - \delta P_i$ are normalized for all $\delta \in [0, \delta_0]$. Then, one can decide the existence of (and then compute) some $\delta_1 > 0$ s.t. $[\![D - \delta Q]\!] = \cup_{1\le i\le b} [\![C_i - \delta P_i]\!]$ for all $0 < \delta < \min(\delta_0, \delta_1)$, in polynomial space and in time $O(|\mathcal{C}_0|^{2b} p(|\mathcal{A}|))$, where $p(\cdot)$ is a polynomial.*

Moreover, in this case, for all shrinking matrices $Q', P'_1, \ldots, P'_b$ s.t. $Q_{x,y} \bowtie (P_i)_{x,y} \Leftrightarrow Q'_{x,y} \bowtie (P'_i)_{x,y}$ and $(P_i)_{x,y} \bowtie (P_j)_{x,y} \Leftrightarrow (P'_i)_{x,y} \bowtie (P'_j)_{x,y}$ for all $i, j \in \{1, \ldots, b\}$, $x, y \in \mathcal{C}_0$ and $\bowtie \in \{<, =\}$, it holds $[\![D - \delta Q']\!] = \cup_{1\le i\le b} [\![C_i - \delta P'_i]\!]$ for all small enough $\delta > 0$.

Note that checking equality between a zone and a union of zones is a difficult problem; some heuristics were suggested in [13]. The second point of the lemma says that the satisfaction of the left part of (6) by a shrunk solution only depends on the relative ordering of the

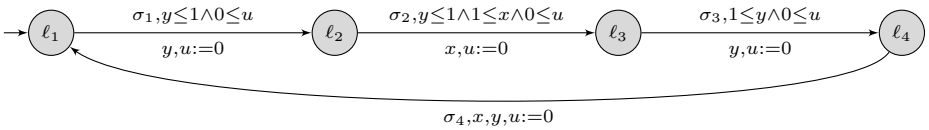


Figure 3 A shrinkable timed automaton. One can in fact shrink guards g_{σ_i} into $g'_{\sigma_1} = 3\delta \leq x \wedge y \leq 1 - \delta \wedge y - x \leq 1 - 4\delta \wedge f(\delta)$, $g'_{\sigma_2} = 1 + \delta \leq x \wedge y \leq 1 - 2\delta \wedge 3\delta \leq x - y \wedge f(\delta)$ and $g'_{\sigma_3} = y \leq 1 - \delta \wedge f(\delta)$, where $f(\delta) = \delta \leq u \wedge y - u \leq 1 - 2\delta$, The resulting shrunk automaton can be seen to be non-blocking and time-abstract similar to $\mathcal{A}$ for any $\delta \in [0, \frac{1}{4}]$. Notice how additional constraints appear in the guards.

coefficients of the shrinking matrices. Therefore, we only need to guess the ordering between all parameters (there is at least one if there exists a shrunk solution), and solve the right part of (6) augmented with these guessed (in)equalities.

Formally, let Φ be the max-plus equation corresponding to the right part of (6), as given by Lemma 9. Let $\mathbf{k}'$ denote the set of all parameters that appear in Φ (there is one parameter per element of each matrix G_σ and $M_{\sigma,\sigma'}$). Notice that $\mathbf{k}'$ has size $O\big((|\mathcal{C}_0| \cdot |\mathcal{L}| \cdot b)^2\big)$, where b is the maximal number of outgoing edges in $\mathcal{A}$, and that Φ has size polynomial in the size of $\mathcal{A}$. Φ is a conjunction of equations $k = \phi_k(\mathbf{k}')$ for all $k \in \mathbf{k}'$. For all pairs $k, l \in \mathbf{k}'$, we guess a relation among $\{<, =, >\}$, and define equation Φ' by adding these relations to Φ. This can be done, for the case $k = l$, by replacing the constraints on k and l respectively by $k = \max(\phi_k(\mathbf{k}'), l)$ and $l = \max(\phi_k(\mathbf{k}'), k)$, and in the case $k > l$, by replacing the constraint on k by $k = \max(\phi_k(\mathbf{k}'), l + 1)$. Notice that Φ' is obtained from Φ in polynomial time and with a polynomial number of guesses. We then solve Φ' using Theorem 7. If we find a solution, say $(P_\sigma)_\sigma \cup (P_{\sigma,\sigma'})_{\sigma,\sigma'}$, we verify that $[\![G_\sigma - \delta P_\sigma]\!] = \cup_{\sigma'} [\![M_{\sigma,\sigma'} - \delta P_{\sigma,\sigma'}]\!]$ for small δ, for all pairs $(\sigma, \sigma') \in \Sigma_{E \circ E}$, in time $O(|\mathcal{C}_0|^{2b} p(|\mathcal{A}|))$ and in polynomial space by Lemma 10. We accept if all verifications succeed and reject otherwise. If accepted, any solution provides a shrunk solution of (6), by Lemma 9. Conversely, if there is a shrunk solution of (6), then, Φ' can be constructed for the guesses corresponding to this solution, and by Lemma 10, Φ' has a solution. If b is fixed, this procedure is in NP. Otherwise, instead of making guesses, we can deterministically try all possible guesses (the number of possible guesses is $O(2^{(|\mathcal{C}| \cdot |\mathcal{L}| \cdot b)^2})$ and verify in polynomial space, so the procedure is then in PSPACE.

Finally, to decide shrinkability, one can first compute parameters $\mathbf{k}$ and δ_0 for non-blockingness, then check shrinkability w.r.t. simulation since the latter does not depend on $\mathbf{k}$ and δ_0. Figure 3 shows an example of a shrinkable timed automaton.

6 Implementation Semantics

In this section, we present an *implementation semantics*, which takes into account reaction times and clock imprecisions. Our semantics corresponds to the execution of timed automata by a digital system that has a single digital clock and nonzero reaction time. Our semantics is closely related to the one studied in [15] with minor differences, but we prove additional properties besides the one given there. We first define our semantics and state its properties, then compare it with [15], and with other related work.

We describe a system which interacts, via sending and receiving signals, with a physical environment (*e.g.* via sensors). We distinguish input and output actions, and define the transitions of the system taking into account the imprecisions of the clock, the transmission delay of signals and the reaction time of the system. When an event is generated at time T by the environment, it is treated by the system at time $T + \epsilon$, for some $\epsilon > 0$ which will be

bounded but unpredictable. Similarly, when the environment receives a signal at time T, it must have been sent at some time $T - \epsilon$. We assume that the system ignores any signal that is received during the treatment of the previous signal; this *reaction time* will be also bounded but unpredictable. We define the timestamps of both input and output actions as the reaction times of the environment, since we are interested in the behaviour of the environment controlled by a digital timed system.

The implementation semantics has three parameters: a) Δ_c is the clock period, b) Δ_r is the maximum *reaction time*, following each action, c) Δ_t is the maximum *transmission delay* of signals between the system and the environment (ϵ above). We suppose the system has a Δ_c-periodic clock, whose value, at any real time T, is $\lfloor T \rfloor_{\Delta_c} = \max_{k \geq 0}\{k\Delta_c \mid k\Delta_c \leq T\}$.

▶ **Definition 11.** Let $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ be a TA with $\Sigma = \Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}}$, and $\Delta_r, \Delta_c, \Delta_t > 0$. The *implementation semantics* $[\![\mathcal{A}]\!]^{\mathsf{Impl}}$ is the TTS $(S_{\mathcal{A}}, s_0, \Sigma, E)$ in which states are tuples (ℓ, T, v, u_0): ℓ is a location, $T \in \mathbb{R}_{\geq 0}$ the current real time, $v \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$ the timestamp of the latest reset for each clock, and $u_0 \in [0, \Delta_r]$ the reaction time following the latest location change. From any state (ℓ, T, v, u_0), for any edge $\ell \xrightarrow{\sigma, g, R} \ell'$ and $T' \geq T$, we let,

- if $\sigma \in \Sigma_{\mathsf{in}}$, $(\ell, T, v, u_0) \xrightarrow{\sigma(T')} (l', T'+\epsilon, v[R \leftarrow T'+\epsilon], u'_0)$, whenever $\lfloor T'+\epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models g$ and $T' + \epsilon \geq T + u_0$, where $(\epsilon, u'_0) \in [0, \Delta_t] \times [0, \Delta_r]$ is chosen non-deterministically,
- if $\sigma \in \Sigma_{\mathsf{out}}$, $(\ell, T, v, u_0) \xrightarrow{\sigma(T')} (\ell', T', v[R \leftarrow T'-\epsilon], u'_0)$, whenever $\lfloor T'-\epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models g$, $\epsilon < (T' - T)$, and $T' - \epsilon \geq T + u_0$, where $(\epsilon, u'_0) \in [0, \Delta_t] \times [0, \Delta_r]$ is chosen non-deterministically.

Notice that ϵ and u_0 are bounded by known values but are unpredictable, so they cannot be chosen by the system. We will consider *scheduler* functions ρ, which, depending on the history of a given run, chooses (ϵ, u_0) at each transition. For any scheduler ρ, we denote by $[\![\mathcal{A}]\!]^{\mathsf{Impl}}_{\rho}$ the implementation semantics, where (ϵ, u_0) is given by ρ at each transition. We will not formally define ρ here, but it can be done without difficulty.

The following proposition states the relation between the exact semantics and the implementation semantics of timed automata. All properties hold under *any* scheduler ρ. For any TTS $\mathcal{T}$, let us write $\mathcal{T}^{\geq \alpha}$, the TTS obtained from $\mathcal{T}$ where consecutive transitions are separated by at least α time units.

▶ **Proposition 12.** *Let $\mathcal{A}$ be a timed automaton s.t. $[\![\mathcal{A}]\!]$ is non-blocking, and $\Delta_r, \Delta_c, \Delta_t > 0$. Then, for any $\Delta \geq 2\Delta_c + 4\Delta_t + \Delta_r$ and scheduler ρ, $[\![\mathcal{A}_\Delta]\!]^{\mathsf{Impl}}_{\rho}$ is non-blocking and,*

$$[\![\mathcal{A}]\!]^{\geq 2\Delta_r + \Delta_t} \sqsubseteq [\![\mathcal{A}_\Delta]\!]^{\mathsf{Impl}}_{\rho} \sqsubseteq [\![\mathcal{A}_{\Delta + 2\Delta_c + 4\Delta_t}]\!].$$

Now, for any timed automaton $\mathcal{A}$, consider $\mathcal{A}'$ as defined in Section 3.2. We have

$$[\![\mathcal{A}'_{-\mathbf{k}\delta}]\!] \quad \sqsubseteq \quad [\![\mathcal{A}'_{-\mathbf{k}\delta+\Delta}]\!]^{\mathsf{Impl}}_{\rho} \quad \sqsubseteq \quad [\![\mathcal{A}'_{-\mathbf{k}\delta+\Delta'}]\!] \quad \sqsubseteq \quad [\![\mathcal{A}']\!] = [\![\mathcal{A}]\!],$$

whenever $\Delta \geq 2\Delta_c + 4\Delta_t + \Delta_r$, $\Delta' = \Delta + 2\Delta_c + 4\Delta_t$ and $\delta \geq \max(2\Delta_r + \Delta_t, \Delta')$. In fact, $[\![\mathcal{A}'_{-\mathbf{k}\delta}]\!]^{\geq 2\Delta_r + \Delta_t}$ is equal to $[\![\mathcal{A}'_{-\mathbf{k}\delta}]\!]$ whenever $\delta \geq 2\Delta_r + \Delta_t$, and the rightmost simulation is due to the fact that $-\mathbf{k}\delta + \Delta' < 0$. Thus, given appropriate parameters, the implementation semantics of a shrunk automaton is always a refinement of the exact semantics of the original automaton. Moreover, when $\mathcal{A}'$ is shrinkable (say, with parameters $\mathbf{k}\delta$), then $[\![\mathcal{A}'_{-\mathbf{k}\delta+\Delta}]\!]^{\mathsf{Impl}}_{\rho}$ is also non-blocking and $[\![\mathcal{A}']\!] \sqsubseteq_{\text{t.a.}} [\![\mathcal{A}'_{-\mathbf{k}\delta+\Delta}]\!]^{\mathsf{Impl}}_{\rho}$. Thus, the properties of shrinkable timed automata are preserved in implementation.

6.1 Related Work

A similar semantics, called the *program semantics*, was defined in [15] and was proven to be simulated by the enlarged semantics (as in the rightmost simulation in Proposition 12). Our

definition follows their ideas, but the main difference is that our semantics is not *input-enabled*, that is, it can ignore signals during the treatment of another signal, and has no buffer. Both assumptions are applicable to different platforms (see [4, 16] for examples of systems that ignore any signal unless it is maintained long enough). Moreover, instead of detailing the reception and the treatment of the signals in several steps, we rather define action transitions taking a positive unpredictable amount of time, during which computations take place. This allows us to model the unpredictability using schedulers and state our properties *for any scheduler*. Note that two results in Proposition 12 are new compared to [15]: the leftmost simulation and the preservation of non-blockingness. Another recent work considers the behaviour of timed automata when actions have long execution times [1] but it assumes perfect clocks. A different line of work considers the implementability of timed automata extended with tasks but imprecisions and reaction times are not considered [19].

References

1 T. Abdellatif, J. Combaz, and J. Sifakis. Model-based implementation of real-time applications. In EMSOFT'10, p. 229–238, New York, NY, USA, 2010. ACM.

2 K. Altisen and S. Tripakis. Implementation of timed automata: An issue of semantics or modeling? In FORMATS'05, LNCS 3829, p. 273–288. Springer, 2005.

3 R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

4 E. Asarin, O. Maler, and A. Pnueli. On discretization of delays in timed automata and digital circuits. In CONCUR'98, LNCS 1466, p. 470–484. Springer, 1998.

5 F. Baccelli et al. *Synchronization and Linearity – An Algebra For Discrete Event Systems*. John Wiley & Sons, 1992.

6 J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, LNCS 2098, p. 87–124. Springer, 2004.

7 G. Berry. The foundations of Esterel. In *Proof, Language, and Interaction – Essays in Honour of Robin Milner*, p. 425–454. MIT Press, 2000.

8 P. Bouyer et al. Timed automata can always be made implementable. In CONCUR'11, LNCS 6901, p. 76–91, Aachen, Germany, 2011. Springer.

9 P. Bouyer, N. Markey, and P.-A. Reynier. Robust model-checking of linear-time properties in timed automata. In LATIN'06, LNCS 3887, p. 238–249. Springer, 2006.

10 P. Bouyer, N. Markey, and P.-A. Reynier. Robust analysis of timed automata via channel machines. In FoSSaCS'08, LNCS 4962, p. 157–171. Springer, 2008.

11 F. Cassez, T. A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In HSCC'02, LNCS 2289, p. 134–148. Springer, 2002.

12 P. Chamuczyński. *Algorithms and data structures for parametric analysis of real time systems*. PhD thesis, University of Göttingen, Germany, 2009.

13 A. David et al. Model checking timed automata with priorities using DBM subtraction. In FORMATS'06, LNCS 4202, p. 128–142. Springer, 2006.

14 M. De Wulf et al. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, 2008.

15 M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.

16 H. Dierks. PLC-automata: a new class of implementable real-time automata. *Theoretical Computer Science*, 253:61–93, 2001.

17 D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In AVMFSS'89, LNCS 407, p. 197–212. Springer, 1990.

18 M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In FOCS'95, p. 453–462, 1995.

19 T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. In EMSOFT'01, LNCS 2211, p. 166–184. Springer, 2001.

20 T. Hune et al. Linear parametric model checking of timed automata. In TACAS'01, LNCS 2031, p. 189–203. Springer, 2001.

21 R. Jaubert and P.-A. Reynier. Quantitative robustness analysis of flat timed automata. In FOSSACS'11, LNCS 6604, p. 229–244. Springer, 2011.

22 A. Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

23 O. Sankur. Untimed language preservation in timed systems. In MFCS'11, LNCS 6907, p. 556–567, Warsaw, Poland, 2011. Springer.

24 A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

The Quantitative Linear-Time–Branching-Time Spectrum

Uli Fahrenberg[1], Axel Legay[1], and Claus Thrane[2]

1 **INRIA/IRISA, Campus de Beaulieu, 35042 Rennes CEDEX, France**
2 **Department of Computer Science, Aalborg University, 9220 Aalborg Øst, Denmark**

Abstract

We present a distance-agnostic approach to quantitative verification. Taking as input an unspecified distance on system traces, or executions, we develop a game-based framework which allows us to define a spectrum of different interesting system distances corresponding to the given trace distance. Thus we extend the classic linear-time–branching-time spectrum to a quantitative setting, parametrized by trace distance. We also provide fixed-point characterizations of all system distances, and we prove a general transfer principle which allows us to transfer counterexamples from the qualitative to the quantitative setting, showing that all system distances are mutually topologically inequivalent.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Quantitative verification, System distance, Distance hierarchy, Linear time, Branching time

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.103

1 Introduction

For rigorous design and verification of embedded systems, both qualitative and quantitative information and constraints have to be taken into account [16, 18, 20]. This applies to the *models* considered, to the *properties* one wishes to be satisfied, and to the *verification* itself. Hence the question asked in quantitative verification is not "Does the system satisfy the requirements?", but rather "*To which extent* does the system satisfy the requirements?" Standard qualitative verification techniques are inherently *fragile*: either the requirements are satisfied, or they are not, regardless of how close the actual system might come to the specification. To overcome this lack of robustness, notions of *distance* between systems are essential.

As pointed out in [16], qualitative and quantitative aspects of verification should be treated orthogonally in any theory of quantitative verification (of course they can hardly be separated in practice, but that is not of our concern here). The formalism we propose in this paper addresses this orthogonality by modeling qualitative aspects using standard *labeled transition systems* and expressing the quantitative aspects using *trace distances*, or distances on system executions. Based on these ingredients, we develop a comprehensive theory of *system distances* which generalizes the standard linear-time–branching-time spectrum [12, 13, 24] to a quantitative setting, see Figure 1.

Similarly to [3], our theory relies on Ehrenfeucht-Fraïssé games and allows for a more refined analysis of systems. More precisely, our parametrized framework forms a hierarchy of games, for each trace distance used in its instantiation. In the quantitative setting, using games with real-valued outcomes, as opposed to discrete games, effectively allows us obtain

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 103–114

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

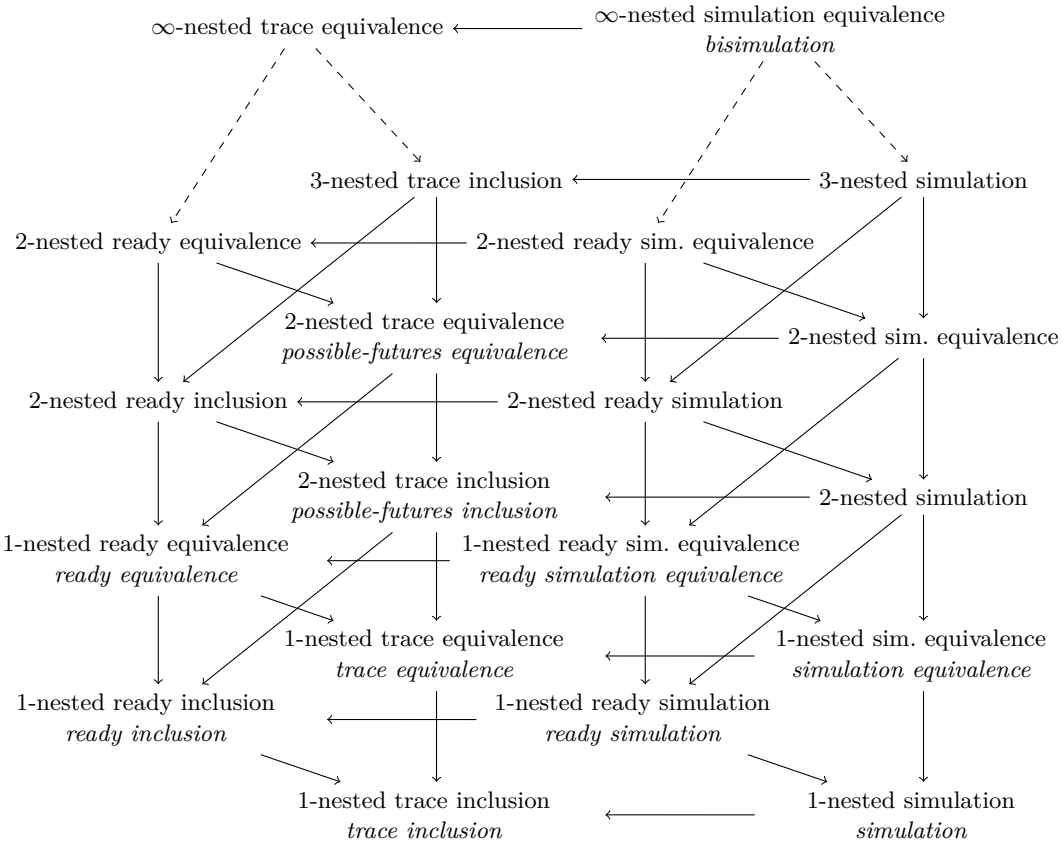


Figure 1 Parts of the quantitative linear-time–branching-time spectrum. The nodes are the different system distances introduced in this paper, and an edge $d_1 \longrightarrow d_2$ indicates that $d_1(s,t) \geq d_2(s,t)$ for all states s, t, and that d_1 and d_2 are topologically inequivalent.

a continuous verdict on the relationship between systems, and hence to detect the difference between minor and major discrepancies between systems.

Indeed the view of this paper is that in a theory of quantitative verification, the quantitative aspects should be treated just as much as an input to a verification problem as the qualitative aspects are. Hence it is of limited use to develop a theory pertaining only to some *specific* quantitative measures like the ones in [1, 2, 4, 10, 17, 22, 23] and other papers which all treat only a few specific ways of measuring distances; any theory of quantitative verification should work just as well regardless of the way the engineers decide to measure differences between system executions.

We take as input a labeled transition system and a trace distance; both are unspecified except for some general characteristic properties. Based on this information and using the theory of *quantitative games*, we lift most of the linear-time–branching-time spectrum of van Glabbeek [24] to the quantitative setting, while the rest may be obtained in a similar way using minor additional conditions as described in [3]. We show that all the distinct equivalences in van Glabbeek's spectrum correspond to topologically inequivalent distances in the quantitative setting.

As our framework is independent of the chosen trace distance, we are essentially adding a second, quantitative, dimension to the linear-time–branching-time spectrum. In this terminology, the first dimension is the qualitative one which concerns the different linear and

branching ways of specifying qualitative constraints, and the second dimension bridges the gap between the trivial van-Glabbeek spectrum in which everything is equivalent, and the discrete spectrum in which everything is fragile.

The authors wish to thank Luca Aceto for some insightful comments on a previous version of this paper. Note that due to space constraints, some proofs had to be omitted.

2 Traces, Trace Distances, and Transition Systems

In this paper, the set $\mathbb{N}$ of natural numbers includes 0; the set of positive natural numbers is denoted by $\mathbb{N}_+$. For a finite non-empty sequence $a = (a_0, \dots, a_n)$, we write $\mathrm{last}(a) = a_n$ and $\mathsf{len}(a) = n+1$ for the length of a; for an infinite sequence a we let $\mathsf{len}(a) = \infty$. Concatenation of finite sequences a and b is denoted $a \cdot b$. We denote by $a^k = (a_k, a_{k+1}, \dots)$ and a_i the k-shift, and ith element respectively, of a (finite or infinite) sequence, and by ϵ the empty sequence.

A function $d : X \times X \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ on a set X is called a *hemimetric* if $d(x,x) = 0$ and $d(x,y) + d(y,z) \geq d(x,z)$ for all $x, y, z \in X$. If d is such that $d(x,y) = 0$ implies $x = y$ for all $x, y \in X$, it is called a *quasimetric*. Two hemimetrics d_1 and d_2 on a set X are said to be *topologically equivalent* if the topologies on X generated by the open balls $B_i(x;r) = \{y \in X \mid d_i(x,y) < r\}$, for $i = 1, 2$, $x \in X$, and $r > 0$, coincide. Topological equivalence hence preserves topological notions such as convergence of sequences: If a sequence (x_j) of points in X converges in one hemimetric, then it also converges in the other. As a consequence, topological equivalence of d_1 and d_2 implies that for all $x, y \in X$, $d_1(x,y) = 0$ if and only if $d_2(x,y) = 0$.

Topological equivalence is the weakest of the common notions of equivalence for metrics; it does not preserve metric properties such as distances or angles. We are hence mainly interested in topological equivalence as a tool for showing negative properties; we will later prove a number of results on topological *inequivalence* of metrics which imply that any other reasonable metric equivalence also fails for these cases.

Throughout this paper we fix a set $\mathbb{K}$ of labels, and we let $\mathbb{K}^\infty = \mathbb{K}^* \cup \mathbb{K}^\omega$ denote the set of finite and infinite traces (*i.e.* sequences) in $\mathbb{K}$. A hemimetric $d^T : \mathbb{K}^\infty \times \mathbb{K}^\infty \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is called a *trace distance* if $\mathsf{len}(\sigma) \neq \mathsf{len}(\tau)$ implies $d^T(\sigma, \tau) = \infty$. (Note that we hence apply the *asymmetric* view on distances as *e.g.* in [4].)

A *labeled transition system* (LTS) is a pair (S, T) consisting of states S and transitions $T \subseteq S \times \mathbb{K} \times S$. We often write $s \xrightarrow{x} t$ to signify that $(s, x, t) \in T$. Given $e = (s, x, t) \in T$, we write $\mathit{src}(e) = s$, $\mathit{tgt}(e) = t$ for the source and target of e. For a (finite or infinite) path π in a LTS we denote by $\mathrm{tr}(\pi) \in \mathbb{K}^\infty$ the trace induced by π. For $s \in S$ we denote by $\mathrm{Pa}(s)$ the set of (finite or infinite) paths from s and by $\mathrm{Tr}(s) = \{\mathrm{tr}(\pi) \mid \pi \in \mathrm{Pa}(s)\}$ the set of traces from s.

2.1 Examples of Trace Distances

We show here a number of trace distances with which our quantitative framework can be instantiated. Note that each such distance gives rise to its own linear-time–branching-time spectrum in the quantitative dimension.

Most of the trace distances one finds in the literature are defined by giving a distance on labels in $\mathbb{K}$ and a method to combine these distances on individual symbols to a distance on traces. Three general methods are used for this combination:

- the *point-wise* trace distance: $\mathsf{PW}_\lambda(d)(\sigma,\tau) = \sup_j \lambda^j d(\sigma_j, \tau_j)$;
- the *accumulating* trace distance: $\mathsf{ACC}_\lambda(d)(\sigma,\tau) = \sum_j \lambda^j d(\sigma_j, \tau_j)$;
- the *limit-average* trace distance: $\mathsf{AVG}(d)(\sigma,\tau) = \liminf_j \frac{1}{j+1} \sum_{i=0}^{j} d(\sigma_i, \tau_i)$.

Here λ is a *discounting* factor with $0 < \lambda \leq 1$, and we assume that the involved traces have equal length; otherwise any trace distance has value ∞. Note that all trace distances are parametrized by the label distance d. The point-wise distance thus measures the (discounted) greatest individual symbol distance in the traces, whereas accumulating and limit-average distance accumulate these individual distances along the traces.

If the label distance d on $\mathbb{K}$ is the *discrete* distance given by $d_{\mathrm{disc}}(x,x) = 0$ and $d_{\mathrm{disc}}(x,y) = \infty$ for $x \neq y$, then all trace distances above agree, for any λ. This defines the *discrete trace distance* $d^T_{\mathrm{disc}} = \mathsf{PW}_\lambda(d_{\mathrm{disc}}) = \mathsf{ACC}_\lambda(d_{\mathrm{disc}}) = \mathsf{AVG}(d_{\mathrm{disc}})$ given by $d^T_{\mathrm{disc}}(\sigma,\tau) = 0$ if $\sigma_j = \tau_j$ for all j, and ∞ otherwise. We will show below that for the discrete trace distance, our quantitative linear-time–branching-time spectrum specializes to the qualitative one of [24].

If one lets $d(x,x) = 0$ and $d(x,y) = 1$ for $x \neq y$ instead, then $\mathsf{ACC}_1(d)$ is *Hamming distance* [14] for finite traces, and $\mathsf{ACC}_\lambda(d)$ with $\lambda < 1$ and $\mathsf{AVG}(d)$ are two sensible ways to define Hamming distance also for infinite traces. $\mathsf{PW}_1(d)$ is topologically equivalent to the discrete distance — $\mathsf{PW}_1(d)(\sigma,\tau) = 1$ if and only if $d^T_{\mathrm{disc}}(\sigma,\tau) = \infty$.

Point-wise and accumulating distances (for concrete instances of label distances d and concrete instantiations of $\mathbb{K}$) have been studied in a number of papers [1, 2, 4, 10, 17, 22, 23]. $\mathsf{PW}_1(d)$ is the point-wise distance from [4,6,10,17,22], and $\mathsf{PW}_\lambda(d)$ for $\lambda < 1$ is the discounted distance from [4, 5]. Accumulating distance $\mathsf{ACC}_\lambda(d)$ has been studied in [10, 17, 22], and $\mathsf{AVG}(d)$ *e.g.* in [1,2]. Both $\mathsf{ACC}_\lambda(d)$ and $\mathsf{AVG}(d)$ are well-known from the theory of discounted and mean-payoff games [9, 25].

All distances above were obtained from distances on individual symbols in $\mathbb{K}$. A trace distance for which this is *not* the case is the *maximum-lead* distance from [15, 22] defined for $\mathbb{K} \subseteq \Sigma \times \mathbb{R}$. Writing $x \in \mathbb{K}$ as $x = (x^\ell, x^w)$, it is given by

$$d^T_{\pm}(\sigma,\tau) = \begin{cases} \sup_j \left| \sum_{i=0}^{j} \sigma_i^w - \sum_{i=0}^{j} \tau_i^w \right| & \text{if } \sigma_j^\ell = \tau_j^\ell \text{ for all } j, \\ \infty & \text{otherwise.} \end{cases}$$

As a last example of a trace distance we mention the *Cantor* distance given by $d^T_{\mathrm{C}}(\sigma,\tau) = (1 + \inf\{j \mid \sigma_j \neq \tau_j\})^{-1}$. Cantor distance hence measures the (inverse) length of the common prefix of the sequences and has been used for verification *e.g.* in [7]. Both Hamming and Cantor distance have applications in information theory and pattern matching.

We will return to our example trace distances in Section 5.2 to show how our framework may be applied to yield concrete formulations of distances in the linear-time–branching-time spectrum relative to these.

3 Quantitative Ehrenfeucht-Fraïssé Games

To lift the linear-time–branching-time spectrum to the quantitative setting, we define below a quantitative Ehrenfeucht-Fraïssé game [8, 11] on a given LTS (S, T) which is similar to the game hierarchy in [3] and the well-known bisimulation game of [21]. The intuition of the game is as follows: The two players, with Player 1 starting the game, alternate to choose transitions, or *moves*, in T, starting with transitions from given start states s and t and continuing their choices from the targets of the transitions chosen in the previous step. At each of his turns, Player 1 also makes a choice whether to choose a transition from the target of his own previous choice, or from the target of his opponent's previous choice (to "switch paths"). We use a *switch counter* to keep track of how often Player 1 has chosen to switch paths. Player 2 has then to respond with a transition from the remaining target. This game is played for an infinite number of rounds, or until one player runs out of choices, thus

building two finite or infinite paths. The value of the game is then the trace distance of the traces of these two paths.

A Player-1 *configuration* of the game is a tuple $(\pi, \rho, m) \in T^n \times T^n \times \mathbb{N}$, for $n \in \mathbb{N}$, such that for all $i \in \{0, \dots, n-2\}$, either $src(\pi_{i+1}) = tgt(\pi_i)$ and $src(\rho_{i+1}) = tgt(\rho_i)$, or $src(\pi_{i+1}) = tgt(\rho_i)$ and $src(\rho_{i+1}) = tgt(\pi_i)$. Similarly, a Player-2 configuration is a tuple $(\pi, \rho, m) \in T^{n+1} \times T^n \times \mathbb{N}$ such that for all $i \in \{0, \dots, n-2\}$, either $src(\pi_{i+1}) = tgt(\pi_i)$ and $src(\rho_{i+1}) = tgt(\rho_i)$, or $src(\pi_{i+1}) = tgt(\rho_i)$ and $src(\rho_{i+1}) = tgt(\pi_i)$; and $src(\pi_n) = tgt(\pi_{n-1})$ or $src(\pi_n) = tgt(\rho_{n-1})$. The set of all Player-$i$ configurations is denoted Conf_i.

Intuitively, the configuration (π, ρ, m) keeps track of the history of the game; π stores the choices of Player 1, ρ the choices of Player 2, and m is the switch counter. Hence π and ρ are sequences of transitions in T which can be arranged by suitable swapping to form two paths $(\bar\pi,\bar\rho)$. How exactly these sequences are constructed is determined by a pair of *strategies* which specify for each player which edge to play from any configuration.

A Player-1 strategy is hence a partial mapping $\theta_1 : \mathsf{Conf}_1 \to T \times \mathbb{N}$ such that for all $(\pi, \rho, m) \in \mathsf{Conf}_1$ for which $\theta_1(\pi, \rho, m) = (e', m')$ is defined,

- $src(e') = tgt(\mathrm{last}(\pi))$ and $m' = m$ or $m' = m + 1$, or
- $src(e') = tgt(\mathrm{last}(\rho))$ and $m' = m + 1$.

A Player-2 strategy is a partial mapping $\theta_2 : \mathsf{Conf}_2 \to T \times \mathbb{N}$ such that for all $(\pi \cdot e, \rho, m) \in \mathsf{Conf}_2$ for which $\theta_2(\pi \cdot e, \rho, m) = (e', m')$ is defined, $m' = m$, and $src(e') = tgt(\mathrm{last}(\rho))$ if $src(e) = tgt(\mathrm{last}(\pi))$, $src(e') = tgt(\mathrm{last}(\pi))$ if $src(e) = tgt(\mathrm{last}(\rho))$. The sets of Player-1 and Player-2 strategies are denoted Θ_1 and Θ_2. Note that we only allow Player 1 to switch paths if he also increases the switch counter.

We can now define what it means to *update* a configuration according to a strategy: For $\theta_1 \in \Theta_1$ and $(\pi, \rho, m) \in \mathsf{Conf}_1$, $\mathsf{upd}_{\theta_1}(\pi, \rho, m)$ is defined if $\theta_1(\pi, \rho, m) = (e', m')$ is defined, and then $\mathsf{upd}_{\theta_1}(\pi, \rho, m) = (\pi \cdot e', \rho, m')$. Similarly, for $\theta_2 \in \Theta_2$ and $(\pi \cdot e, \rho, m) \in \mathsf{Conf}_2$, $\mathsf{upd}_{\theta_2}(\pi \cdot e, \rho, m)$ is defined if $\theta_2(\pi \cdot e, \rho, m) = (e', m')$ is defined, and then $\mathsf{upd}_{\theta_2}(\pi \cdot e, \rho, m) = (\pi \cdot e, \rho \cdot e', m')$.

A pair of states $(s, t) \in S \times S$ and a pair of strategies $(\theta_1, \theta_2) \in \Theta_1 \times \Theta_2$ inductively determine a sequence (π^j, ρ^j, m^j) of configurations given by $(\pi^0, \rho^0, m^0) = (s, t, 0)$, $(\pi^{2j+1}, \rho^{2j+1}, m^{2j+1}) = \mathsf{upd}_{\theta_1}(\pi^{2j}, \rho^{2j}, m^{2j})$ and $(\pi^{2j}, \rho^{2j}, m^{2j}) = \mathsf{upd}_{\theta_2}(\pi^{2j-1}, \rho^{2j-1}, m^{2j-1})$; the sequence is understood to finish as soon as one of the updates is undefined.

The configurations in this sequence satisfy $\pi^j \sqsubseteq \pi^{j+1}$, $\rho^j \sqsubseteq \rho^{j+1}$ for all j, where $\sqsubseteq$ denotes prefix ordering, hence the limits $\pi = \varinjlim \pi^j$, $\rho = \varinjlim \rho^j$ exist (as potentially infinite paths). By our conditions on configurations, the pair (π, ρ) in turn determines a pair $(\bar\pi, \bar\rho)$ of *paths* in S, as follows:

$$(\bar\pi_1, \bar\rho_1) = \begin{cases} (\pi_1, \rho_1) & \text{if } src(\pi_1) = s \\ (\rho_1, \pi_1) & \text{if } src(\pi_1) = t \end{cases} \qquad (\bar\pi_j, \bar\rho_j) = \begin{cases} (\pi_j, \rho_j) & \text{if } src(\pi_j) = tgt(\bar\pi_{j-1}) \\ (\rho_j, \pi_j) & \text{if } src(\pi_j) = tgt(\bar\rho_{j-1}) \end{cases}$$

The *outcome* of the game when played from (s, t) according to a strategy pair (θ_1, θ_2) is $\mathsf{out}(\theta_1, \theta_2)(s, t) = (\bar\pi, \bar\rho)$, and its *utility* is $\mathsf{util}(\theta_1, \theta_2)(s, t) = d^T(\mathrm{tr}(\mathsf{out}(\theta_1, \theta_2)(s, t))) = d^T(\mathrm{tr}(\bar\pi), \mathrm{tr}(\bar\rho))$, where d^T is given as a parameter to the game. The objective of Player 1 in the game is to maximize utility, whereas Player 2 wants to minimize it. Hence we define the *value* of the game from (s, t) to be

$$v(s, t) = \sup_{\theta_1 \in \Theta_1} \inf_{\theta_2 \in \Theta_2} \mathsf{util}(\theta_1, \theta_2)(s, t).$$

For a given subset $\Theta_1' \subseteq \Theta_1$ we will write

$$v(\Theta_1')(s, t) = \sup_{\theta_1 \in \Theta_1'} \inf_{\theta_2 \in \Theta_2} \mathsf{util}(\theta_1, \theta_2)(s, t),$$

and if we need to emphasize dependency of the value on the given trace distance, we write $v(d^T, \Theta_1')$. The following lemma states the immediate fact that if Player 1 has fewer strategies available, the game value decreases.

▶ **Lemma 1.** *For all $\Theta_1' \subseteq \Theta_1'' \subseteq \Theta_1$ and all $s, t \in S$, $v(\Theta_1')(s,t) \leq v(\Theta_1'')(s,t)$.*

In the following we will need two technical conditions on strategies and on trace distances. We say that a strategy $\theta_1 \in \Theta_1$ is *uniform* if it holds for all configurations $(\pi, \rho, m), (\pi, \rho', m), (\pi', \rho, m) \in \mathsf{Conf}_1$ that whenever $\theta_1(\pi, \rho, m) = (e', m')$ is defined,

- if $src(e') = tgt(\pi)$, then also $\theta_1(\pi, \rho', m)$ is defined, and
- if $src(e') = tgt(\rho)$, then also $\theta_1(\pi', \rho, m)$ is defined.

Uniformity of strategies is used to combine paths built from different starting states in the proof of Proposition 2 below. A subset $\Theta_1' \subseteq \Theta_1$ is uniform if all strategies in Θ_1' are uniform; the concrete strategy subsets we will consider in later sections will all be uniform.

We say that a trace distance d^T is *well-behaved* if $\sup_{\theta_1 \in \Theta_1} \inf_{\theta_2 \in \Theta_2} \mathsf{util}(\theta_1, \theta_2)(s,t) = \inf_{\theta_2 \in \Theta_2} \sup_{\theta_1 \in \Theta_1} \mathsf{util}(\theta_1, \theta_2)(s,t)$ for all $s, t \in S$. This assumption is related to determinacy of the quantitative path-building game, asserting that each pair of states *has* a value, and ultimately to determinacy of Gale-Steward games [19]. Note that if it holds, then so does the same equation with Θ_1 replaced by any subset $\Theta_1' \subseteq \Theta_1$.

We finish this section by showing that under certain conditions, the game value is a distance, and that results concerning inequalities in the qualitative dimension can be transfered to topological inequivalences in the quantitative setting. Say that a Player-1 strategy $\theta_1 \in \Theta_1$ is *non-switching* if it holds for all (π, ρ, m) for which $\theta_1(\pi, \rho, m) = (e', m')$ is defined that $m = m'$, and let Θ_1^0 be the set of non-switching Player-1 strategies. The following proposition shows that for any uniform strategy subset which contains all non-switching strategies and any well-behaved trace distance, the value of our quantitative game is a hemimetric.

▶ **Proposition 2.** *If $\Theta_1' \subseteq \Theta_1$ is uniform and $\Theta_1^0 \subseteq \Theta_1'$, and if d^T is well-behaved, then $v(\Theta_1')$ is a hemimetric on S.*

Next we show a powerful *transfer principle* which allows us to generalize counterexamples regarding the equivalences in the qualitative linear-time–branching-time spectrum [24] to the qualitative setting. We will make use of this principle later to show that all distances we introduce are topologically inequivalent.

▶ **Theorem 3** (Transfer principle)**.** *Assume the LTS (S, T) to be finitely branching and d^T to be a well-behaved quasimetric, and let $\Theta_1', \Theta_1'' \subseteq \Theta_1$. If there exist $s, t \in S$ for which $v(d^T_{\mathrm{disc}}, \Theta_1')(s,t) = 0$ and $v(d^T_{\mathrm{disc}}, \Theta_1'')(s,t) = \infty$, then $v(d^T, \Theta_1')$ and $v(d^T, \Theta_1'')$ are topologically inequivalent.*

Proof. By $v(d^T_{\mathrm{disc}}, \Theta_1')(s,t) = 0$, and as (S, T) is finitely branching, we know that for any $\theta_1 \in \Theta_1'$ there exists $\theta_2 \in \Theta_2$ for which $(\bar{\pi}, \bar{\rho}) = \mathsf{out}(\theta_1, \theta_2)(s,t)$ satisfy $\mathrm{tr}(\bar{\pi}) = \mathrm{tr}(\bar{\rho})$, hence also $v(d^T, \Theta_1')(s,t) = 0$. Conversely, and as d^T is a quasimetric, $v(d^T, \Theta_1'')(s,t) = 0$ would imply that also $v(d^T_{\mathrm{disc}}, \Theta_1'')(s,t) = 0$, hence we must have $v(d^T, \Theta_1'')(s,t) \neq 0$, entailing topological inequivalence. ◀

4 The Distance Spectrum

In this section we introduce the distances depicted in Figure 1 and show their mutual relationship. Note again that the results obtained here are independent of the particular

trace distance considered; in the terminology of the introduction we are developing a linear-time–branching-time spectrum at every point of the quantitative dimension. In order to capture the remaining equivalences in the original spectrum [24], we may easily adopt the approach from [3] which imposes one of three extra conditions which Player 1 may choose to invoke and thereby terminate the game.

Let $(S, T \subseteq S \times \mathbb{K} \times S)$ be a LTS and $d^T : \mathbb{K}^\infty \times \mathbb{K}^\infty \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ a trace distance.

4.1 Branching Distances

If the switching counter in the game introduced in Section 3 is unbounded, Player 1 can choose at any move whether to prolong the previous choice or to switch paths, hence this resembles the bisimulation game [21].

▶ **Definition 4.** The *bisimulation distance* between s and t is $d^{\text{bisim}}(s,t) = v(s,t)$.

Note again that bisimulation distance, and indeed all distances defined in this section, are parametrized by the trace distance d^T.

▶ **Theorem 5.** *For $d^T = d^T_{\text{disc}}$ the discrete trace distance, $d^{\text{bisim}}_{\text{disc}}(s,t) = 0$ iff s and t are bisimilar.*

Proof. By discreteness of d^T_{disc}, we have $d^{\text{bisim}}_{\text{disc}}(s,t) = 0$ if and only if it holds that for all $\theta_1 \in \Theta_1$ there exists $\theta_2 \in \Theta_2$ for which $\mathsf{util}(\theta_1, \theta_2)(s,t) = 0$. Hence for each reachable Player-1 configuration (π, ρ, m) with $\theta_1(\pi, \rho, m) = (e', m')$, we have $\theta_2(\pi \cdot e', \rho, m') = (e'', m')$ with $\mathrm{tr}(e') = \mathrm{tr}(e'')$, *i.e.* Player 2 matches the labels chosen by Player 1 precisely, implying that s and t are bisimilar. The proof of the other direction is trivial. ◀

We can restrict the strategies available to Player 1 by allowing only a pre-defined finite number of switches:

$$\Theta_1^{k\text{-sim}} = \{\theta_1 \in \Theta_1 \mid \text{if } \theta_1(\pi, \rho, m) = (e', m') \text{ is defined, then } m' \leq k-1\}$$

In the so-defined k-nested simulation game, Player 1 is only allowed to switch paths $k-1$ times during the game. Note that $\Theta_1^{1\text{-sim}} = \Theta_1^0$ is the set of non-switching strategies.

▶ **Definition 6.** The *k-nested simulation distance* from s to t, for $k \in \mathbb{N}_+$, is $d^{k\text{-sim}}(s,t) = v(\Theta_1^{k\text{-sim}})(s,t)$. The *$k$-nested simulation equivalence distance* between s and t is $d^{k\text{-sim-eq}}(s,t) = \max(v(\Theta_1^{k\text{-sim}})(s,t), v(\Theta_1^{k\text{-sim}})(t,s))$.

▶ **Theorem 7.** *For $d^T = d^T_{\text{disc}}$ the discrete trace distance,*

- *$d^{k\text{-sim}}_{\text{disc}}(s,t) = 0$ iff there is a k-nested simulation from s to t,*
- *$d^{k\text{-sim-eq}}_{\text{disc}}(s,t) = 0$ iff there is a k-nested simulation equivalence between s and t.*

Especially, $d^{1\text{-sim}}_{\text{disc}}$ corresponds to the usual *simulation* preorder, and $d^{2\text{-sim}}_{\text{disc}}$ to *nested simulation*. Similarly, $d^{1\text{-sim-eq}}_{\text{disc}}$ is *similarity*, and $d^{2\text{-sim-eq}}_{\text{disc}}$ is *nested similarity*. The proof is similar to the one of Theorem 5.

▶ **Theorem 8.** *For all $k, \ell \in \mathbb{N}_+$ with $k < \ell$ and all $s, t \in S$,*

$$d^{k\text{-sim-eq}}(s,t) \leq d^{\ell\text{-sim}}(s,t) \leq d^{\ell\text{-sim-eq}}(s,t) \leq d^{\text{bisim}}(s,t).$$

If the trace distance d^T is a well-behaved quasimetric and the LTS (S,T) is finitely branching, then all distances above are topologically inequivalent.

Proof. The first part of the theorem follows from $\Theta_1^{k\text{-sim-eq}} \subseteq \Theta_1^{\ell\text{-sim}} \subseteq \Theta_1^{\ell\text{-sim-eq}} \subseteq \Theta_1$ and Lemma 1. Topological inequivalence follows from Theorem 3 and the fact that for the discrete relations corresponding to the distances above (obtained by letting $d^T = d^T_{\text{disc}}$), the inequalities are strict [24]. ◀

As a variation of k-nested simulation, we can consider strategies which allow Player 1 to switch paths k times during the game, but after the last switch, he may only pose *one* transition as a challenge, to which Player 2 must answer, and then the game finishes:

$$\Theta_1^{k\text{-rsim}} = \{\theta_1 \in \Theta_1 \mid \text{if } \theta_1(\pi,\rho,m) \text{ is defined, then } m \leq k-1\}$$

▶ **Definition 9.** The *k-nested ready simulation distance* from s to t, for $k \in \mathbb{N}_+$, is $d^{k\text{-rsim}}(s,t) = v(\Theta_1^{k\text{-rsim}})(s,t)$. The *$k$-nested ready simulation equivalence distance* between s and t is $d^{k\text{-rsim-eq}}(s,t) = \max(v_1(\Theta_1^{k\text{-rsim}})(s,t), v_1(\Theta_1^{k\text{-rsim}})(t,s))$.

For the discrete case, only $k = 1$ seems to have been considered; the proof is again similar to the one of Theorem 5.

▶ **Theorem 10.** *For $d^T = d^T_{\text{disc}}$ the discrete trace distance,*
- *$d^{1\text{-rsim}}_{\text{disc}}(s,t) = 0$ iff there is a ready simulation from s to t,*
- *$d^{1\text{-rsim-eq}}_{\text{disc}}(s,t) = 0$ iff s and t are ready simulation equivalent.*

The next theorem finishes our work on the right half of Figure 1; its proof is similar to the one of Theorem 8.

▶ **Theorem 11.** *For all $k, \ell \in \mathbb{N}_+$ with $k < \ell$ and all $s, t \in S$,*

$$d^{k\text{-sim}}(s,t) \leq d^{k\text{-rsim}}(s,t) \leq d^{\ell\text{-sim}}(s,t), \qquad d^{k\text{-sim-eq}}(s,t) \leq d^{k\text{-rsim-eq}}(s,t) \leq d^{\ell\text{-sim-eq}}(s,t).$$

Additionally, $d^{k\text{-rsim}}$ and $d^{k\text{-sim-eq}}$ are incomparable, and also $d^{k\text{-rsim-eq}}$ and $d^{(k+1)\text{-sim}}$ are incomparable. If the trace distance d^T is a well-behaved quasimetric and the LTS (S,T) is finitely branching, then all distances above are topologically inequivalent.

4.2 Linear Distances

Above we have introduced the distances in the right half of the quantitative linear-time–branching-time spectrum in Figure 1 and shown the relations claimed in the diagram. To develop the left half, we need the notion of *blind* strategies. For any subset $\Theta_1' \subseteq \Theta_1$ we define the set of blind Θ_1'-strategies by

$$\tilde{\Theta}_1' = \{\theta_1 \in \Theta_1' \mid \forall \pi,\rho,\rho',m : \theta_1(\pi,\rho,m) = \theta_1(\pi,\rho',m), \\ \text{or } \theta_1(\pi,\rho,m) = (e,m+1) \text{ and } \mathit{tgt}(\mathrm{last}(\rho)) \neq \mathit{tgt}(\mathrm{last}(\rho'))\}.$$

Hence in such a blind strategy, either the edge chosen by Player 1 does not depend on the choices of Player 2, or the switch counter is increased, in which case the Player-1 choice only depends on the target of the last choice of Player 2. Now we can define, for $s, t \in S$ and $k \in \mathbb{N}_+$,
- the *∞-nested trace equivalence distance*: $d^{\infty\text{-trace-eq}}(s,t) = v_1(\tilde{\Theta}_1)(s,t)$,
- the *k-nested trace distance*: $d^{k\text{-trace}}(s,t) = v_1(\tilde{\Theta}_1^{k\text{-sim}})(s,t)$,
- the *k-nested trace equivalence distance*:
$$d^{k\text{-trace-eq}}(s,t) = \max(v_1(\tilde{\Theta}_1^{k\text{-sim}})(s,t), v_1(\tilde{\Theta}_1^{k\text{-sim}})(t,s)),$$
- the *k-nested ready distance*: $d^{k\text{-ready}}(s,t) = v_1(\tilde{\Theta}_1^{k\text{-rsim}})(s,t)$, and

- the *k-nested ready equivalence distance*:
$$d^{k\text{-ready-eq}}(s,t) = \max(v_1(\tilde{\Theta}_1^{k\text{-rsim}})(s,t), v_1(\tilde{\Theta}_1^{k\text{-rsim}})(t,s)).$$

Using the discrete trace distance, we recover the following standard relations [24].

▶ **Theorem 12.** *For $d^T = d^T_{\mathrm{disc}}$ the discrete trace distance and $s,t \in S$ we have*
- $d^{1\text{-trace}}_{\mathrm{disc}}(s,t) = 0$ *iff there is a trace inclusion from s to t,*
- $d^{1\text{-trace-eq}}_{\mathrm{disc}}(s,t) = 0$ *iff s and t are trace equivalent,*
- $d^{2\text{-trace}}_{\mathrm{disc}}(s,t) = 0$ *iff there is a possible-futures inclusion from s to t,*
- $d^{2\text{-trace-eq}}_{\mathrm{disc}}(s,t) = 0$ *iff s and t are possible-futures equivalent,*
- $d^{1\text{-ready}}_{\mathrm{disc}}(s,t) = 0$ *iff there is a readiness inclusion from s to t,*
- $d^{1\text{-ready-eq}}_{\mathrm{disc}}(s,t) = 0$ *iff s and t are ready equivalent.*

The following theorem entails all relations in the left side of Figure 1; the right-to-left arrows follow from the strategy set inclusions $\tilde{\Theta}'_1 \subseteq \Theta'_1$ for any $\Theta'_1 \subseteq \Theta_1$ and Lemma 1. As with Theorems 8 and 11, the theorem follows by strategy set inclusion, Theorem 3, and corresponding results for the discrete relations.

▶ **Theorem 13.** *For all $k,\ell \in \mathbb{N}_+$ with $k < \ell$ and $s,t \in S$,*

$$d^{k\text{-trace-eq}}(s,t) \le d^{\ell\text{-trace}}(s,t) \le d^{\ell\text{-trace-eq}}(s,t) \le d^{\infty\text{-trace-eq}}(s,t),$$
$$d^{k\text{-trace}}(s,t) \le d^{k\text{-ready}}(s,t) \le d^{\ell\text{-trace}}(s,t),$$
$$d^{k\text{-trace-eq}}(s,t) \le d^{k\text{-ready-eq}}(s,t) \le d^{\ell\text{-trace-eq}}(s,t).$$

Additionally, $d^{k\text{-ready}}$ and $d^{k\text{-trace-eq}}$ are incomparable, and also $d^{k\text{-ready-eq}}$ and $d^{(k+1)\text{-trace}}$ are incomparable. If the trace distance d^T is a well-behaved quasimetric and the LTS (S,T) is finitely branching, then all distances above are topologically inequivalent.

5 Recursive Characterizations

Now we turn our attention to an important special case in which the given trace distance has a specific recursive characterization; we show that, in this case, all distances in the spectrum can be characterized as least fixed points. We will see in Section 5.2 that this can be applied to all examples of trace distances mentioned in Section 2.1. Note that all theorems require the LTS in question to be finitely branching; this is a standard assumption which goes back to [21] and is also necessary in our case.

5.1 Fixed-Point Characterizations

Let L be a complete lattice with order $\sqsubseteq$ and bottom and top elements $\bot$, $\top$. Let $f : \mathbb{K}^\infty \times \mathbb{K}^\infty \to L$, $g : L \to \mathbb{R}_{\ge 0} \cup \{\infty\}$, $F : \mathbb{K} \times \mathbb{K} \times L \to L$ such that $d^T = g \circ f$, g is monotone, $F(x,y,\cdot) : L \to L$ is monotone for all $x,y \in \mathbb{K}$, and

$$f(\sigma,\tau) = \begin{cases} F(\sigma_0,\tau_0,f(\sigma^1,\tau^1)) & \text{if } \sigma,\tau \neq \boldsymbol{\epsilon}, \\ \top & \text{if } \sigma = \boldsymbol{\epsilon}, \tau \neq \boldsymbol{\epsilon} \text{ or } \sigma \neq \boldsymbol{\epsilon}, \tau = \boldsymbol{\epsilon}, \\ \bot & \text{if } \sigma = \tau = \boldsymbol{\epsilon} \end{cases} \tag{1}$$

for all $\sigma,\tau \in \mathbb{K}^\infty$.

We hence assume that d^T has a recursive characterization (using F) on top of an arbitrary lattice L which we introduce between $\mathbb{K}^\infty$ and $\mathbb{R}_{\ge 0} \cup \{\infty\}$ to serve as a *memory*.

▶ **Theorem 14.** *The endofunction I on $(\mathbb{N}_+ \cup \{\infty\}) \times \{1,2\} \to L^{S\times S}$ defined by*

$$I(h_{m,p})(s,t) = \begin{cases} \max \begin{cases} \sup\limits_{s \xrightarrow{x} s'} \inf\limits_{t \xrightarrow{y} t'} F(x,y,h_{m,1}(s',t')) \\ \sup\limits_{t \xrightarrow{y} t'} \inf\limits_{s \xrightarrow{x} s'} F(x,y,h_{m-1,2}(s',t')) \end{cases} & \text{if } m \geq 2, p = 1 \\ \sup\limits_{s \xrightarrow{x} s'} \inf\limits_{t \xrightarrow{y} t'} F(x,y,h_{m,1}(s',t')) & \text{if } m = 1, p = 1 \\ \max \begin{cases} \sup\limits_{t \xrightarrow{y} t'} \inf\limits_{s \xrightarrow{x} s'} F(x,y,h_{m,2}(s',t')) \\ \sup\limits_{s \xrightarrow{x} s'} \inf\limits_{t \xrightarrow{y} t'} F(x,y,h_{m-1,1}(s',t')) \end{cases} & \text{if } m \geq 2, p = 2 \\ \sup\limits_{t \xrightarrow{y} t'} \inf\limits_{st \xrightarrow{x} s'} F(x,y,h_{m,2}(s',t')) & \text{if } m = 1, p = 2 \end{cases}$$

has a least fixed point $h^* : (\mathbb{N}_+ \cup \{\infty\}) \times \{1,2\} \to L^{S\times S}$, and if the LTS (S,T) is finitely branching, then $d^{k\text{-sim}} = g \circ h^*_{k,1}$, $d^{k\text{-sim-eq}} = g \circ \max(h^*_{k,1}, h^*_{k,2})$ for all $k \in \mathbb{N}_+ \cup \{\infty\}$.

Hence I iterates the function h over the branching structure of (S,T), computing all nested branching distances at the same time. Note the specialization of this to simulation distance, where we have the following fixed-point equation, using $h^*_{1,1} = h^{1\text{-sim}}$:

$$h^{1\text{-sim}}(s,t) = \sup_{s \xrightarrow{x} s'} \inf_{t \xrightarrow{y} t'} F(x,y,h^{1\text{-sim}}(s',t'))$$

An equally compact expression may be derived for bisimulation distance, and similar theorems for all the other distances in the quantitative linear-time–branching-time spectrum can also be derived.

The fixed-point characterizations above immediately lead to iterative *algorithms* for computing the respective distances: to compute *e.g.* simulation distance, we can initialize $h^{1\text{-sim}}(s,t) = 0$ for all states $s,t \in S$ and then iteratively apply the above equality. This assumes the LTS (S,T) to be finitely branching and uses Kleene's fixed-point theorem and continuity of F. Note however that this computation is only guaranteed to converge to simulation distance in finitely many steps in case the lattice $L^{S\times S}$ is *finite*.

5.2 Recursive Characterizations for Example Distances

We show that the considerations in Section 5.1 apply to all the example distances we introduced in Section 2.1. We apply Theorem 14 to derive fixed-point formulas for corresponding simulation distances, but of course all other distances in the quantitative linear-time–branching-time spectrum have similar characterizations.

Let d be a hemimetric on $\mathbb{K}$, then for all $\sigma, \tau \in \mathbb{K}^\infty$ and $0 < \lambda \leq 1$,

$$\mathsf{ACC}_\lambda(d)(\sigma,\tau) = \begin{cases} d(\sigma_0,\tau_0) + \lambda\, \mathsf{ACC}_\lambda(d)(\sigma^1,\tau^1)) & \text{if } \sigma,\tau \neq \boldsymbol{\epsilon}, \\ \infty & \text{if } \sigma = \boldsymbol{\epsilon}, \tau \neq \boldsymbol{\epsilon} \text{ or } \sigma \neq \boldsymbol{\epsilon}, \tau = \boldsymbol{\epsilon}, \\ 0 & \text{if } \sigma = \tau = \boldsymbol{\epsilon}, \end{cases}$$

hence we can apply the iteration theorems with lattice $L = \mathbb{R}_{\geq 0} \cup \{\infty\}$, $g = \mathrm{id}$ the identity function, and the recursion function F given like the formulas above. Using Theorem 14 we can *e.g.* derive the following fixed-point expression for simulation distance:

$$\mathsf{ACC}_\lambda(d)^{1\text{-sim}}(s,t) = \sup_{s \xrightarrow{x} s'} \inf_{t \xrightarrow{y} t'} (d(x,y) + \lambda\, \mathsf{ACC}_\lambda(d)^{1\text{-sim}}(s',t'))$$

Similar considerations apply to the point-wise distances, with "+" replaced by "max". Incidentally, these are exactly the expressions introduced, in an ad-hoc manner, in [4, 10, 22].

Also note that if S is finite with $|S| = n$, then undiscounted point-wise distance $\mathsf{PW}_1(d)$ can only take on the finitely many values $\{d(x,y) \mid (s,x,s'),(t,y,t') \in T\}$, hence the fixed-point algorithm given by Kleene's theorem converges in at most n^2 steps. This algorithm is used in [4, 6, 17]. For undiscounted accumulating distance $\mathsf{ACC}_1(d)$, it can be shown [17] that with $D = \max\{d(x,y) \mid (s,x,s'),(t,y,t') \in T\}$, distance is either infinite or bounded above by $2n^2D$, hence also here the algorithm either converges in at most $2n^2D$ steps or diverges.

For the limit-average distance $\mathsf{AVG}(d)$, we let $L = (\mathbb{R}_{\geq 0} \cup \{\infty\})^{\mathbb{N}}$, $g(h) = \liminf_j h(j)$, and $f(\sigma,\tau)(j) = \frac{1}{j+1}\sum_{i=0}^{j} d(\sigma_i,\tau_i)$ the j-th average. The intuition is that L is used for "remembering" how long in the traces we have progressed with the computation. With F given by $F(x,y,h)(n) = \frac{1}{n+1}d(x,y) + \frac{n}{n+1}h(n-1)$ it can be shown that (1) holds, giving the following fixed-point expression for limit-average simulation distance:

$$h_n^{1\text{-sim}}(s,t) = \sup_{s \xrightarrow{x} s'} \inf_{t \xrightarrow{y} t'} \left(\tfrac{1}{n+1}d(x,y) + \tfrac{n}{n+1}h_{n-1}^{1\text{-sim}}(s',t')\right)$$

For the maximum-lead distance, we let $L = (\mathbb{R}_{\geq 0} \cup \{\infty\})^{\mathbb{R}}$, the lattice of mappings from leads to maximum leads. Using the notation from Section 2.1, we let $g(h) = h(0)$ and $f(\sigma,\tau)(\delta) = \max(|\delta|, \sup_j |\delta + \sum_{i=0}^{j}\sigma_i^w - \sum_{i=0}^{j}\tau_j^w|)$ the maximum-lead distance between σ and τ assuming that σ already has a lead of δ over τ. With $F(x,y,h)(\delta) = \max(|\delta + x - y|, h(\delta + x - y))$ it can be shown that (1) holds, and then the fixed-point expression for maximum-lead simulation distance becomes the one given in [15]:

$$h^{1\text{-sim}}(\delta)(s,t) = \sup_{s \xrightarrow{x} s'} \inf_{t \xrightarrow{y} t'} \max(|\delta + x - y|, h^{1\text{-sim}}(s',t')(\delta + x - y))$$

It can be shown [15] that for S finite with $|S| = n$ and $D = \max\{d(x,y) \mid (s,x,s'),(t,y,t') \in T\}$, the iterative algorithm for computing maximum-lead distance either converges in at most $2n^2D$ steps or diverges.

Regarding Cantor distance, a useful recursive formulation is $f(\sigma,\tau)(n) = f(\sigma^1,\tau^1)(n+1)$ if $\sigma_0 = \tau_0$ and n otherwise, which iteratively counts the number of matching symbols in σ and τ. Here we use $L = (\mathbb{R}_{\geq 0} \cup \{\infty\})^{\mathbb{N}}$, and $g(h) = \frac{1}{h(0)}$; note that the order on L has to be reversed for g to be monotone. The fixed-point expression for Cantor simulation distance becomes

$$h_n^{1\text{-sim}}(s,t) = \max(n, \sup_{s \xrightarrow{x} s'} \inf_{t \xrightarrow{x} t'} h_{n+1}^{1\text{-sim}}(s',t'))$$

but as the order on L is reversed, the sup now means that Player 1 is trying to *minimize* this expression, and Player 2 tries to maximize it. Hence Player 2 tries to find maximal matching *subtrees*; the corresponding Cantor simulation equivalence distance between s and t hence is the inverse of the maximum depth of matching subtrees under s and t.

References

1. Pavol Černý, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. In *CONCUR*, volume 6269 of *LNCS*, pages 253–268. Springer, 2010.
2. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4), 2010.
3. Xin Chen and Yuxin Deng. Game characterizations of process equivalences. In *APLAS*, volume 5356 of *LNCS*, pages 107–121. Springer, 2008.

4 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Transactions on Software Engineering*, 35(2):258–273, 2009.

5 Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In *ICALP*, volume 2719 of *LNCS*, pages 1022–1037. Springer, 2003.

6 Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate analysis of probabilistic processes. In *QEST*, pages 264–273. IEEE Computer Society, 2008.

7 Laurent Doyen, Thomas A. Henzinger, Axel Legay, and Dejan Ničković. Robustness of sequential circuits. In *ACSD*, pages 77–84. IEEE Computer Society, 2010.

8 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961.

9 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

10 Uli Fahrenberg, Kim G. Larsen, and Claus Thrane. A quantitative characterization of weighted Kripke structures in temporal logic. *Computing and Informatics*, 29(6+):1311–1324, 2010.

11 Roland Fraïssé. Sur quelques classifications des systèmes de relations. *Publications Scientifiques de l'Université d'Alger, Série A*, 1:35–182, 1954.

12 David de Frutos Escrig and Carlos Gregorio Rodríguez. (Bi)simulations up-to characterise process semantics. *Information and Computation*, 207(2):146–170, 2009.

13 David de Frutos Escrig, Carlos Gregorio Rodríguez, and Miguel Palomino. On the unification of process semantics: Equational semantics. In *MFPS*, volume 249 of *ENTCS*, pages 243–267. Elsevier, 2009.

14 Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950.

15 Thomas A. Henzinger, Rupak Majumdar, and Vinayak Prabhu. Quantifying similarities between timed systems. In *FORMATS*, volume 3829 of *LNCS*, pages 226–241. Springer, 2005.

16 Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In *FM*, volume 4085 of *LNCS*, pages 1–15. Springer, 2006.

17 Kim G. Larsen, Uli Fahrenberg, and Claus Thrane. Metrics for weighted transition systems: Axiomatization and complexity. *Theoretical Computer Science*, 412(28):3358–3369, 2011.

18 Edward A. Lee. Absolutely positively on time: What would it take? *IEEE Computer*, 38(7):85–87, 2005.

19 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

20 John A. Stankovic, Insup Lee, Aloysius K. Mok, and Raj Rajkumar. Opportunities and obligations for physical computing systems. *IEEE Computer*, 38(11):23–31, 2005.

21 Colin Stirling. Modal and temporal logics for processes. In *Banff Higher Order Workshop*, volume 1043 of *LNCS*, pages 149–237. Springer, 1995.

22 Claus Thrane, Uli Fahrenberg, and Kim G. Larsen. Quantitative analysis of weighted transition systems. *Journal of Logic and Algebraic Programming*, 79(7):689–703, 2010.

23 Franck van Breugel. A behavioural pseudometric for metric labelled transition systems. In *CONCUR*, volume 3653 of *LNCS*, pages 141–155. Springer, 2005.

24 Rob J. van Glabbeek. The linear time – branching time spectrum I. In *Handbook of Process Algebra*, Chapter 1, pages 3–99. Elsevier, 2001.

25 Uri Zwick and Michael Paterson. The complexity of mean payoff games. In *Computing and Combinatorics*, volume 959 of *LNCS*, pages 1–10. Springer, 1995.

Isomorphism testing of read-once functions and polynomials

Raghavendra Rao B.V.[1] and Jayalal Sarma M.N.[2]

1 **Department of Computer Science, Saarland University**
bvrr@cs.uni-saarland.de
2 **Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India**
jayalal@cse.iitm.ac.in

Abstract

In this paper, we study the isomorphism testing problem of formulas in the Boolean and arithmetic settings. We show that isomorphism testing of Boolean formulas in which a variable is read at most once (known as read-once formulas) is complete for log-space. In contrast, we observe that the problem becomes polynomial time equivalent to the graph isomorphism problem, when the input formulas can be represented as OR of two or more monotone read-once formulas. This classifies the complexity of the problem in terms of the number of reads, as read-3 formula isomorphism problem is hard for coNP.

We address the polynomial isomorphism problem, a special case of polynomial equivalence problem which in turn is important from a cryptographic perspective [19, 16]. As our main result, we propose a deterministic polynomial time canonization scheme for polynomials computed by constant-free read-once arithmetic formulas. In contrast, we show that when the arithmetic formula is allowed to read a variable twice, this problem is as hard as the graph isomorphism problem.

1998 ACM Subject Classification F.2.1 [Numerical Algorithms and Problems] – Computations on polynomials, F.1.3 [Complexity Measures and Classes], F.2.3 [Tradeoffs between Complexity Measures]

Keywords and phrases Computational Complexity, Boolean Isomorphism, Read Once Polynomials

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.115

1 Introduction

Computational isomorphism problems between various mathematical structures has intriguing computational complexity (see [6] for a survey). An important example, the Graph Isomorphism(GI) problem asks : given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ decide if there is a bijection $\sigma : V_1 \to V_2$ such that $(u, v) \in E_1 \iff (\sigma(u), \sigma(v)) \in E_2$. This study becomes more important when the structures are computational models by themselves. Checking equivalence between programs is undecidable in general, but has useful special cases with respect to other computational models. We consider isomorphism testing of two important computational structures: Boolean and arithmetic circuits.

A Boolean formula (also known as an expression) is a natural non-uniform model of computing a Boolean function. The corresponding isomorphism question is to decide if the given boolean functions (input as formulas) are equivalent via a bijective transformation of the variables. This problem is known as the Formula isomorphism (FI for short) problem

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 115–126

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the literature. In general FI is in Σ_2^P (*i.e.* , the second level of the polynomial hierarchy), and unlikely to be Σ_2^P-hard unless the polynomial hierarchy collapses to the third level [2]. Goldsmith *et al.* [13] showed that FI for monotone formulas is as hard as general case. (See also [7, 11] for more results on the structure of FI.) Though it can be easily seen that FI is coNP-hard, an exact complexity characterization for FI is unknown to date.

This situation motivates one to look for special cases of FI that admit efficient algorithms. The number of reads of each variable in the formula is a restriction. A formula ϕ is not satisfiable if and only if it is isomorphic to the constant formula 0. By duplicating variables and introducing appropriate equivalence clauses, it follows that even when the number of reads is bounded by 3, FI(in CNF form) is coNP-hard.

We now address the intermediate cases; that is when the number of reads is bounded by 1 and 2 respectively. The first case, also known as read-once formulas, is a model that has received a lot of attention in the literature in various contexts, *e.g.,*[4] obtained efficient learning algorithms for read-once formulas. We show:

▶ **Theorem 1.** *Formula isomorphism for read-once formulas is complete for deterministic logarithmic space.*

However, the bound above seems to be tight. If we allow variables to be read twice, then FI becomes GI-hard even in the most primitive case:

▶ **Theorem 2.** *Isomorphism testing of OR of two monotone read-once DNF formulas is complete for* GI.

A natural analogue of the formula isomorphism question in the arithmetic world is about polynomials : given two polynomials $p(x_1, x_2, \ldots, x_n)$ and $q(x_1, x_2, \ldots, x_n)$ decide if there is a non-trivial permutation of the variables such that the polynomials are identical under the permutation. We denote this problem by PI. We assume that polynomials are presented in the form of arithmetic circuits in the non-black-box setting.

The polynomial isomorphism problem can also be seen as a special case of the well-studied polynomial equivalence problem (PE for short), where given two polynomials $p(x_1, x_2, \ldots, x_n)$ and $q(x_1, x_2, \ldots, x_n)$, decide if there is a non-singular matrix $A \in \mathbb{F}^{n\times n}$ such that $q(X) = p(AX)$, where $AX = (\sum_j A_{1,j}x_j, \ldots, \sum_j A_{n,j}x_j)$. The equivalence problem has survived intense efforts to give deterministic polynomial time algorithms (See [21, 16]). The lack of progress was explained by a result in [1], which reduces graph isomorphism problem to equivalence testing of cubic polynomials. The polynomial equivalence problem is expected to be very challenging and there are cryptographic schemes which are based on polynomial equivalence problems[19]. More recently Kayal [16, 15] developed efficient randomized algorithms for equivalence testing for several special classes of polynomials. Indeed, in the case of isomorphism problem, the matrix A is restricted to be a permutation matrix. Our next result shows that this specialization does not really simplify the problem when degree is 3, and in fact the polynomial isomorphism problem for a constant degree d polynomial also reduces to that of degree 3 polynomials.

▶ **Theorem 3.** *For any constant d, the polynomial isomorphism problem for degree d polynomials is polynomial time many-one equivalent to testing isomorphism of degree-3 polynomials, which in turn, is polynomial time many-one equivalent to* GI.

This shows that the polynomial isomorphism problem is also likely to be hard even when the polynomial is given explicitly listing down the monomials, and is harder than graph isomorphism problem. In general, we show that the isomorphism problem of polynomials is easier than the equivalence problem (over $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$).

▶ **Theorem 4.** PI *polynomial time many-one reduces to* PE.

A naive algorithm for this problem would be to guess the permutation and then verify whether the polynomials are the same under this permutation, which is an instance of the well-studied polynomial identity testing and can be solved in coRP. Thus the isomorphism testing problem is in MA. Indeed, the problem is also harder than the polynomial identity testing problem, because a polynomial is isomorphic to a zero polynomial if and only if it is identically zero. Thus, derandomizing the above MA algorithm in general to NP will imply circuit lower bounds[14]. From the above discussion it also follows that polynomial isomorphism problem is in NP if and only if polynomial identity testing is in NP. Also, Thierauf [24] showed that if PI(over $\mathbb{Q}$) is NP-hard then PH collapses to Σ_2^p.

This motivates looking at special cases of polynomial isomorphism problem for making progress. A *read-once polynomial* is a polynomial $f(X) \in \mathbb{Z}[x_1, \ldots, x_n]$ that can be computed by a read-once arithmetic formula. Read-once polynomials have been studied in various contexts in the literature. Bshouty et. al [10] developed efficient learning algorithms for read-once polynomials with membership and equivalence queries. (See also [8, 9].) More recently, Shpilka and Volkovich [22, 23] developed deterministic black-box sub exponential time algorithms for identity testing of read-once polynomials. In the non-black-box setting they give a polynomial time algorithm. We show the following for the isomorphism problem (which is harder than identity testing problem) as our main result.

▶ **Theorem 5.** *Isomorphism testing for constant-free read-once polynomials can be done in deterministic polynomial time.*

We then extend this to the case of arbitrary coefficients but still constant-free (see Theorem 14). As in the case of FI, we show that if we allow variables to be read twice then the polynomial isomorphism problem becomes GI-hard(see Theorem 16.). The structure of the rest of the paper is as follows. We introduce the basics and prove Theorem 4 in section 2. We prove Theorem 1 and 2 in section 3, and the main Theorem 5 in section 4.

2 Preliminaries

All the complexity theory notions used in this paper are standard. For more details, reader is referred to any standard complexity theory book. (See *e.g.,* [5].)

A Boolean formula ϕ is a directed acyclic graph, where out-degree of every node is bounded by 1, and the non-leaf nodes are labeled by $\{\vee, \wedge, \neg\}$ and the leaf nodes are labels by $\{x_1, \ldots, x_n, 0, 1\}$, where $x_1, \ldots, x_n$ are Boolean variables. Without loss of generality, we assume that ϕ has at most one node of out-degree zero, called the *output gate* of the formula. Naturally, with every formula ϕ, we can associate a Boolean function $f_\phi : \{0,1\}^n \to \{0,1\}$ defined as the function computed at the output node of the formula. A Boolean circuit is a generalization of formula wherein the out-degree of every node can be unbounded.

An arithmetic circuit C over a ring $\mathbb{F}$, is a directed acyclic graph where the non-leaf nodes are labeled by $\{+, \times\}$, and the leaf nodes are labeled by $\{x_1, \ldots, x_n\} \cup \mathbb{F}$, where $x_1, \ldots, x_n$ are variables that take values from $\mathbb{F}$, where $\mathbb{F}$ is a ring. In this paper we restrict our attention to cases where $\mathbb{F} \in \{\mathbb{Z}, \mathbb{R}, \mathbb{Q}\}$. Naturally, we can associate a polynomial $p_g \in \mathbb{F}[x_1, \ldots, x_n]$ with any gate g of the arithmetic circuit C. The polynomial computed by C is the polynomial associated with the output gate of C. An arithmetic formula is an arithmetic circuit where the out-degree of every node can be at most one.

A *read-once* formula (ROF for short), is a Boolean formula in which every variable x_i appears at most once as a leaf label, *i.e.* , every variable is read at most once. Similarly we

can define read-once arithmetic formulas, *i.e.* , arithmetic formulas where a variable appears at most once. Polynomials computed by read-once arithmetic formulas are also known as *read-once polynomials* (ROPs for short).

A *constant-free* read-once arithmetic formula is a read-once arithmetic formula, where the only allowed leaf labels are x_i or $-x_i$. A constant-free ROP is a polynomial that can be computed by constant-free read-once arithmetic formula. A *general-constant-free* ROP is an ROP computed by arithmetic read-once formulas with the leaves labeled by $a_i x_i$, where $a_i \in \mathbb{Z} \setminus \{0\}$. For computational purposes, we assume that a constant-free ROP is given as a constant-free read-once formula in the input.

Now we define some notations that are used in Section 4. Let $C_1, \ldots, C_k$ denote a collection of ordered tuples. Then $\mathsf{sort}(C_1, \ldots, C_k)$ denotes the lexicographic sorted list of $C_1, \ldots, C_k$. For $k > 0$, Σ_k denotes the set of all permutations of a k element set. Let $S_1, \ldots, S_n \in \{0,1\}$, then $\mathsf{parity}(S_1, \ldots, S_n) \triangleq (\sum_{i=1}^{n} S_i \mod 2)$; and $\mathsf{binary}(S_1, \ldots S_n) \triangleq \sum_{i=1}^{n} S_i 2^{n-i}$. For $a \in \mathbb{Z} \setminus \{0\}$, $\mathsf{sgn}(a) = 1$ if $a < 0$, and $\mathsf{sgn}(a) = 0$ otherwise.

Isomorphism testing problems : We now define the problems we address in the paper.

Formula Isomorphism(FI): Given two Boolean formulas $F_1(x_1, \ldots, x_n)$, and $F_2(x_1, \ldots, x_n)$ on n variables : $X = \{x_1, \ldots, x_n\}$, test if there exists a permutation $\pi \in S_n$, such that the functions computed by $F_1(x_1, \ldots, x_n)$ and $F_2(x_{\pi(1)}, \ldots, x_{\pi(n)})$ are the same.

Polynomial Isomorphism(PI): Given two polynomials $P, Q \in \mathbb{F}[x_1, \ldots, x_n]$, test if there exists a permutation $\pi \in S_n$ such that $P(x_1, \ldots, x_n) = Q(x_{\pi(1)}, \ldots, x_{\pi(n)})$. $\mathrm{PI}_d(\mathbb{F})$ denotes the special case when P, and Q are of degree at most d. A notion related to isomorphism is *canonization*. A *canonical code* for polynomials is a function $\mathcal{C} : \mathbb{F}[x_1, \ldots, x_n] \to \{0,1\}^*$ such that $\mathcal{C}(f) = \mathcal{C}(g)$ if and only if the polynomials f and g are isomorphic.

Polynomial Equivalence(PE): Given two polynomials $P, Q \in \mathbb{F}[x_1, \ldots, x_n]$, test if there is a non-singular matrix $A = (a_{i,j}) \in GL(n, \mathbb{F})$ such that the polynomials $P(x_1, \ldots, x_n) = Q(y_1, \ldots, y_n)$ where $y_1, \ldots, y_n$ are obtained by applying the linear transformation defined by the row-vectors of A, *i.e.* , $y_i = \sum_{j=1}^{n} a_{i,j} x_j$.

In general we assume that the input polynomials are given as arithmetic circuits. PE_d denotes the restriction of PE where the input polynomials are of degree d. (See [21] for a detailed exposition on this problem). The following equivalence was proved in [21].

▶ Proposition 6 ([21, 20]). GI poly time many-one reduces to PE_3.

In general, though PI is a special case of PE where A is restricted to be a permutation matrix, it is unclear a priori whether PI is easier than PE. We give a reduction from PI to PE over $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$, and $\mathbb{C}$, this proves Theorem 4.

Proof of Theorem 4: Let $f(X)$ and $g(X)$ be the two polynomials given as an input instance of PI, where $X = \{x_1, \ldots, x_n\}$. Let $d = \max\{\deg(f), n\}$, $m > \max\{2n, n+d+4\}$, such that $\mathsf{g}cd(m-2n, d+n+4) = 1$, and $X' = X \cup \{y, z\}$. Define

$$f'(X,y,z) \triangleq f(X) + y^{d+1}x_1 \cdots x_n + z^{d+n+2}(x_1 + \cdots + x_n) + z^{d+n+4} + y^{d+1}z^m; \quad \text{and}$$
$$g'(X,y,z) \triangleq g(X) + y^{d+1}x_1 \cdots x_n + z^{d+n+2}(x_1 + \cdots + x_n) + z^{d+n+4} + y^{d+1}z^m$$

Suppose $f(X) \cong g(X)$, then clearly $f'(X') \cong g'(X')$. Suppose $f'(X') = g'(A'X')$ for some non-singular matrix A'. We claim that A' has to be a permutation matrix. By the degree conditions, A' sends y to by , and z to cz, where $c^{d+n+4} = 1$, and $b^{d+1}c^m = 1$. Also

note that y and z both have zero coefficients in $A'x_i$ for all i, by the unique factorization of $x_1 \cdots x_n$. Similarly, for all i, $A'x_i$ cannot have two non-zero coefficients, again by the degrees of y and z, and the unique factorization of $x_1 \cdots x_n$. The only possibility is, A' could be the product of a permutation matrix P and a diagonal matrix D with determinant equal to 1. Let the i th entry in the diagonal D be λ_i. Then, $z^{d+n+2}A'x_i$ will have coefficient $\lambda_i c^{d+n+2}$, but in the target polynomial f', it has coefficient 1, so $\lambda_i = c^2$. This implies $b^{d+1}c^{2n} = 1$, and hence $c^{m-2n} = 1$. As $\mathsf{gcd}(m-2n, d+n+4) = 1$, $c = 1$, and hence $b = 1$, $\lambda_i = 1$ $1 \leq i \leq n$. Thus, $f(X) \cong g(X)$ if and only if the polynomials $f'(X')$ and $g'(X')$ are equivalent. Note that $f'(X')$ can be computed by a circuit of size $s + 4d + 4n + m + 9$, where s is the size of a circuit computing $f(X)$. This completes the proof.

3 Isomorphism testing of Boolean read-once formulas

For a Boolean read-once formula ϕ, let $G(\phi) = (V_\phi, E_\phi)$ denote the formula graph of ϕ as defined in [4], *i.e.* , $V_\phi = \{x_1, \ldots, x_n\}$, and $E_\phi = \{(x_i, x_j) \mid \mathsf{LCA}(x_i, x_j)$ is labeled $\wedge\}$, where $\mathsf{LCA}(x, y)$ denotes the least common ancestor of the leaves labeled x and y in ϕ.

3.1 Logspace characterization : Proof of Theorem 1

Proof. We first argue the upper bound. We argue for the special case of monotone read-once formulas. Let ϕ_1 and ϕ_2 be two minimal monotone read-once formulas. First observe that $G(\phi_1) \cong G(\phi_2) \iff \phi_1 \cong \phi_2$. Let F_1 (resp. F_2) be the minimum read-once formula computing the same function as ϕ_1 (resp. ϕ_2) by merging consecutive gates of the same type into one gate of larger fan-in. Construct two trees T_1 and T_2 from F_1 and F_2 respectively as follows. We describe the construction for T_1. Treat the formula F_1 as a undirected tree with $\wedge$ gates colored as RED, $\vee$ gates colored as BLUE and the leaf nodes colored as GREEN.

▶ Claim 1. $G(\phi_1) \cong G(\phi_2) \iff T_1 \cong T_2$.

Assuming the claim, testing whether $\phi_1 \cong \phi_2$ is equivalent to isomorphism testing of colored trees. As the latter can be done in deterministic logarithmic space [18], it is enough to prove the claim.

Proof of the claim. ($\Rightarrow$) Suppose $G(\phi_1) \cong G(\phi_2)$, and σ be such a bijection between the vertices of $G(\phi_1)$ and $G(\phi_2)$. Fix the corresponding map between the leaves of T_1 and T_2. For any two leaves x, y of T_1, let $\mathsf{LCA}(x, y)$ denote the least common ancestor of x and y in T_1. Colors and degrees of $\mathsf{LCA}(x, y)$ and $\mathsf{LCA}(\sigma(x), \sigma(y))$ are the same. (This follows from the property of the graphs $G(\phi_1)$ and $G(\phi_2)$.) So σ induces a color-preserving isomorphism between T_1 and T_2.

($\Leftarrow$) Let σ be a color preserving isomorphism between T_1 and T_2. Let π denote the corresponding bijection between the leaves of T_1 and T_2 induced by σ. It is sufficient to argue that $G(\phi_1) = \pi(G(\phi_2))$. Consider two variables x and y. As $color(\mathsf{LCA}(x, y)) = color(\mathsf{LCA}(\pi(x), \pi(y)))$, we have $(x, y) \in E(G(\phi_1)) \iff (\pi(x), \pi(y)) \in E(G_2)$. This completes the proof of the Claim. ◀

The argument above can be extended to the non-monotone case by coloring the leaves of T_f (resp. T_g) that correspond to positive literals as YELLOW, and those corresponding to negative literals as RED.

Now we argue the L-hardness. We reduce directed forest reachability (which is known to be L-complete[12]) to FI. Given the instance (G, s, t) of directed forest reachability where the task is to check if there is a directed path from s to t, we construct the formula(F) as

follows. Ignore the incoming edges to s and outgoing edges from t. Replace s with a variable x and label every other leaf node with the constant 1. Replace all intermediate nodes by $\wedge$ gates. Label t as the output node. Since G is a directed forest, F will be a formula and is a read-once formula by construction. Moreover, F will evaluate to x (and hence isomorphic to the trivial formula x) if and only if there is a directed path from s to t. ◀

3.2 Larger Number of Reads : Proof of Theorem 2

Naturally, one could hope to extend theorem 1 to boolean formulas that read a variable at most a constant number of times. Surprisingly, it turns out that if the input formulas are represented as OR of two monotone read-once formulas, then isomorphism testing becomes GI hard.

▶ **Lemma 7.** GI *polynomial time many-one reduces to testing isomorphism of OR of two monotone read-once formulas given in DNF form.*

Proof. The reduction is from GI for bipartite graphs which is as hard as the general GI[17]. For a simple undirected bipartite graph $G = (U, V, E)$, define a formula $\phi(G)$ on variables $\{x_e \mid e \in E\}$ as follows. For every $v \in U \cup V$, $\phi(G)$ contains the term $x_{e_1} \wedge x_{e_2} \wedge \ldots \wedge x_{e_\ell}$ as a minterm, where $e_1, e_2, \ldots, e_\ell$ are the edges that are incident on v in G. *i.e.* ,

$$\phi_G = \bigvee_{v \in U \cup V} \bigwedge_{e \text{ incident on } v} x_e = \Big(\bigvee_{u \in U} \bigwedge_{e \text{ incident on } u} x_e\Big) \vee \Big(\bigvee_{v \in V} \bigwedge_{e \text{ incident on } v} x_e\Big) \tag{1}$$

So $\phi(G)$ can be written as an OR of two monotone read-once formulas. , $G_1 \cong G_2 \iff \phi(G'_1) \cong \phi(G'_2)$. This concludes the proof. ◀

Observe that a monotone boolean formula ϕ given in DNF form, can also be represented as a bipartite graph with vertices of one side corresponding to variables of ϕ and the terms of ϕ as vertices on the other side, edge relations is defined with respect to inclusion. Combined with Lemma 7, this proves Theorem 2.

4 Isomorphism testing of Read-once polynomials

As starting point, observe that the deterministic polynomial identity testing algorithm for read-once formulas [22] gives an NP upper bound for isomorphism testing of read-once polynomials. A natural question is to see if the NP upper bound above can be improved to a polynomial time algorithm. In the following, we provide a polynomial time algorithm for the isomorphism testing of certain special classes of read-once polynomials. We begin with the toy case of monotone read-once polynomials f such that $f(0) = 0$.

▶ **Lemma 8.** *Isomorphism testing of monotone read-once polynomials that can be computed by monotone read-once arithmetic formulas with leaves labeled from $\{x_1, \ldots, x_n\}$, can be done in deterministic logarithmic space.*

Proof. Let f be a monotone read-once polynomial computed by a monotone read-once formula ϕ_f. Without loss of generality assume that ϕ_f is in the minimal form, i.e., inputs of a $\times$ gate are either $+$ gates or variables and that of a $+$ gate are either $\times$ gates or variables. Let $G_f = (V_f, E_f)$ be the undirected graph with $V_f = \{x_1, \ldots, x_n\}$ and $E_f = \{(x_i, x_j) \mid \mathsf{LCA}_{\phi_f}(x_j, x_j) \text{ is a } \times \text{ gate}\}$. By the definition of G_f, a monomial $M = \prod_{j=1}^{k} x_{i_j}$ has coefficient 1 in f if and only if the vertices $x_{i_1}, \ldots, x_{i_k}$ form a maximal clique in G_f. Let T_f denote the underlying (undirected) tree of ϕ_f, where a node corresponding to a $+$ gate is colored BLUE and that corresponding to a $\times$ gate is colored RED.

Let f and g be two monotone read-once formulas as above. Clearly, $f \cong g \iff G_f \cong G_g$. Now we show that $G_f \cong G_g$ if and only if T_f is isomorphic to T_g as a colored tree.

Suppose $T_f \cong T_g$ via a bijection π between the vertices of T_f and T_g. Let σ be the bijection between the leaves of T_g and T_f induced by π. Then $\forall i \neq j$, $\mathsf{LCA}_{\phi_f}(x_i, x_j)$ is a $\times$ gate if and only if $\mathsf{LCA}_{\phi_g}(x_{\sigma(i)}, x_{\sigma(j)})$ is a $\times$ gate. So σ defines an isomorphism between G_f and G_g.

For the converse direction, suppose $f \cong g$. The proof is by induction on the structure of f and g. The base case is when f and g are single variables, in which case the claim follows. There are two cases:

Case 1: f and g can be written uniquely as $f = f_1 + \ldots + f_k$, $g = g_1 + \ldots + g_k$, where $f_i's$ and $g_i's$ cannot be written as sum of two or more variable disjoint monotone ROP's. Then, $f \cong g$ if and only if there is a permutation $\sigma \in \Sigma_k$ such that $f_i \cong g_{\sigma(i)} \iff G_{f_i} \cong G_{g_{\sigma(i)}} \iff T_{f_i} \cong T_{g_{\sigma(j)}}$, where the last equivalence is available from induction. So, $T_f \cong T_g$.

Case 2: $f = f_1 \times \ldots \times f_k$ and $g = g_1 \times \ldots \times g_k$, where $f_i's$ and $g_i's$ cannot be decomposed into products of two or more variable disjoint ROPs. Then, $f \cong g$ implies there is a permutation $\sigma \in \Sigma_k$ such that $f_i \cong g_{\sigma(i)}$. By induction. This implies $T_{f_i} \cong T_{g_{\sigma(i)}}$, which in turn implies $T_f \cong T_g$. Now the algorithm is obvious: given f and g, compute T_f, and T_g. Then test if $T_f \cong T_g$, using the log-space algorithm for testing isomorphism for trees [18]. ◀

Our goal now is to extend Lemma 8 to the case of non-monotone read-once polynomials. Consider a constant-free read-once formula, *i.e.* , a read-once formula where a leaf is labeled from $\{-x_i, x_i\}$ for some i. An obvious approach would be to use Lemma 8 with an additional coloring of -ve terms. Then the two representations: $f = f_1 \times f_2 \times \ldots \times f_k$, and $g = (-f_1) \times (-f_2) \times f_3 \times \ldots \times f_k$ will give rise to two non-isomorphic trees whereas f and g are identical polynomials.

We overcome this by building a canonical code for general constant-free read-once polynomials along the lines of the well-known tree canonization algorithm [3]. Recall that a canonical code for a polynomial is an object that is unique for every isomorphism class. Also, note that efficient computation of canonical code for a class of polynomials implies efficient algorithm for isomorphism testing for that class, though the converse may not be true in general. For ease of exposition, we give details for the case of constant-free ROPs. We first observe some simple structural properties of constant-free read-once polynomials that serves as a foundation for our construction of canonization.

▶ Proposition 9. A constant-free read-once polynomial $f \neq 0$ has the following recursive structure:

- $f = a_i x_i$, where $a \in \{-1, 1\}$; or
- f is of Type-1, i.e., $f(X) = f_1(X_1) + f_2(X_2) + \ldots + f_k(X_k)$ for a unique $k \geq 2$, where $f_i's$ are constant-free variable disjoint read-once polynomials and $X = X_1 \uplus X_2 \uplus \ldots \uplus X_k$. Also, f_i cannot be written as a sum of two or more variable disjoint constant-free ROPs; or
- f is of Type-2, i.e, $f(X) = f_1(X_1) \times f_2(X_2) \times \ldots \times f_t(X_t)$ for a unique $t \geq 2$, where $f_i's$ are constant-free variable disjoint read-once polynomials and $X = X_1 \uplus X_2 \uplus \ldots \uplus X_t$. Also, f_i cannot be written as a product of two or more constant-free variable disjoint ROPs.

The following structural characterization of constant-free ROPs follows from Proposition 9.

▶ **Lemma 10.** *(a) If f, g are constant-free ROPs of Type-1, i.e. , $f = f_1 + \ldots + f_k$, $g = g_1 + \ldots + g_k$, where $f_i s$ and $g_i s$ are constant-free ROPs of Type-2. Then, $f \cong g \iff \exists \sigma \in \Sigma_k \;\; f_i \cong g_{\sigma(i)}$.*

(b) If f and g are constant-free ROPs of Type-2, i.e. , $f = f_1 \times \ldots \times f_k$, and $g = g_1 \times \ldots \times g_k$, where f_is and g_is are constant-free ROPs of Type-1. Then,

$$f \cong g \iff \left\{ \begin{array}{l} \exists \sigma \in \Sigma_k, \text{ and } a_1, \ldots, a_k \in \{-1, 1\} \text{ such that } f_i \cong a_{\sigma(i)} g_{\sigma(i)} \\ \text{and } \ \mathsf{parity}(a_1, \ldots, a_k) = 0. \end{array} \right\}$$

Proof. For (a), suppose $f = f_1 + \ldots + f_k$, and $g = g_1 + \ldots + g_k$. As f_is (resp. g_is) are variable disjoint, there is no cancellation of monomials of f_is in f, *i.e.* , every monomial appearing in f_i also appears in f with the same coefficient as in f_i. Since each of the f_i's (and g_i's) cannot be written as a sum of two or more variable disjoint constant-free ROPs we have (a). For (b), note that converse direction is clear as $f_1, ..., f_k$ are variable disjoint. Suppose $f \cong g$, via a permutation σ of the variables. Then, $\sigma(f) = \sigma(f_1) \times \sigma(f_2) \times \ldots \times \sigma(f_k) = g_1 \times g_2 \times \ldots \times g_k$. Then, as $\sigma(f_1), \ldots, \sigma(f_k)$ are variable disjoint, there is a $\pi \in \Sigma_k$ such that $\sigma(f_i) = g_{\pi(i)}$ or $\sigma(f_i) = -g_{\pi(i)}$, and $\{i \mid \sigma(f_i) = -g_{\pi(i)}\}$ is even. ◀

A canonization for constant-free ROPs

Combining Lemma 10 with the standard canonization for trees [3], we propose a polynomial time canonization scheme for constant-free read-once polynomials.

We start with an informal description of code. As a toy example consider a linear polynomial f with coefficients in $\{-1, 1\}$. Let N_f be the number of variables with -ve coefficients and P_f be those with +ve coefficients. Clearly, a linear polynomial g is isomorphic to f if and only if $P_g = P_f$ and $N_g = N_f$. So, P_f, and N_f are the canonical values of f that are invariant under permutation of variables. Similarly, if $f = \prod_{i=1}^{k} a_i x_i$ with $a_i \in \{-1, 1\}$, then any $g = \prod_{i=1}^{k} b_i x_i$ is isomorphic to f if and only if the parity of the number of negative coefficients of g is equal to that of f. So, the number of variables, and the parity of the number of -ve coefficients would be an invariant set for f under permutations of variables.

By Lemma 10, if f is of Type-1, *i.e.* ,$f = f_1 + \ldots + f_k$, then any constant-free ROP isomorphic to f, will look like a permutation of f_is. So, a canonization of f would be a sorted ordering of those for f_is. If f is of Type-2, *i.e.* ,$f = f_1 \times \ldots \times f_k$, then, the canonization of f should be invariant when an even number of f_is are multiplied by -1. We handle these constraints by building the canonization for f, denoted by $\mathsf{code}(f)$ in a bottom-up fashion depending on the structure of the constant-free read-once arithmetic formula computing f.

For a constant-free read-once formula f, $\mathsf{code}(f)$ is a quadruple (C, P, N, S), where C is a string, $P, N \in \mathbb{N}$, and $S \in \{0, 1\}$. Here C stores information about the read-once polynomials computed by the sub-formulas at the root gate of the arithmetic formula computing f. The values of P, N, and S depend on the type of f (as in Proposition 9). If f is of Type-1, then $S = 0$, and N intuitively represents the number of "negative" polynomials f_i, and $P = k - N$. When f is of Type-2, $P = N = 0$, and S in some sense represents the parity of the number of "negative" polynomials in $f_1, \ldots, f_k$, where $f = f_1 \times \cdots \times f_k$. Here the term "negative" is used in a tentative sense.

Now we formally define code via induction based on the structure of f as given by Proposition 9. Abusing the notation we use the symbol $\emptyset$ also to denote empty string.

We consider the following four base cases:

base case 1: $f = x_i$, then $\mathsf{code}(f) = (\emptyset, 0, 1, 0)$.

base case 2: $f = -x_i$, then $\mathsf{code}(f) = (\emptyset, 1, 0, 0)$.

base case 3: $f = \sum_{i=1}^{k} a_i x_i$, for some $k > 2$, and $a_i \in \{-1, 1\}$. Let $C_i = \mathsf{code}(a_i x_i)$. Let $i_1, \ldots, i_k$ be such that $C_{i_1}, \ldots, C_{i_k}$ represents the lexicographical sorting of $C_1, \ldots, C_k$. Let $S_i = \mathsf{sgn}(a_i)$. Let $N = \mathsf{binary}(S_{i_1}, \ldots, S_{i_k})$, and $P = \mathsf{binary}(\bar{S}_{i_1}, \ldots, \bar{S}_{i_k})$. Then

$$\mathsf{code}(f) \triangleq ((\langle \emptyset, 0, 1 \rangle, \overset{\text{k times}}{\ldots} \langle \emptyset, 0, 1 \rangle), N, P, 0)$$

base case 4: $f = \prod_{i=1}^{k} a_i x_i$, $a_i \in \{-1, 1\}$. Let $S = 1$, if the number of -1's in $a_1, \ldots, a_k$ is odd, and $S = 0$ otherwise. Define

$$\mathsf{code}(f) \stackrel{\triangle}{=} ((\langle \emptyset, 0, 1\rangle, \overbrace{\ldots}^{\text{k times}} \langle \emptyset, 0, 1\rangle), 0, 0, S)$$

Inductively, assume that, $\mathsf{code}(g) = (C, N, P, 0)$ for a constant-free ROP g of Type-1 on at most $n-1$ variables, and $\mathsf{code}(g) = (C, 0, 0, S)$ for a constant-free ROP g of Type-2 in at most $n-1$ variables. Consider a constant-free ROP f on n variables. By Proposition 9, there are two cases

Type 1: Let $f = f_1 + f_2 + \ldots f_k$, where $f_1, \ldots, f_k$ are constant-free ROPs of Type-2. By induction, suppose $\mathsf{code}(f_i) = (C_i, 0, 0, S_i)$. If $f_i = a x_{j_i}$ for some $1 \leq j_i \leq n$, and $a \in \{-1, 1\}$ then we need to take $\mathsf{code}(f_i) = (\langle \emptyset, 0, 1\rangle, 0, 0, \mathsf{sgn}(a))$. Let $\langle C_{i_1}, \ldots, C_{i_k}\rangle = \mathsf{sort}(C_1, \ldots, C_k)$, $N = \mathsf{binary}(S_{i_1}, \ldots, S_{i_k})$, and $P = \mathsf{binary}(\bar{S}_{i_1}, \ldots, \bar{S}_{i_k})$. Then,

$$\mathsf{code}(f) \stackrel{\triangle}{=} (\langle C_{i_1}, \ldots, C_{i_k}\rangle, N, P, 0) \tag{2}$$

Type 2: $f = f_1 \times f_2 \times \ldots \times f_k$, where $f_1, \ldots, f_k$ are constant-free ROPs of Type-1. By induction, suppose $\mathsf{code}(f_i) = (C_i, N_i, P_i, 0)$. Let $N_i' = \min\{N_i, P_i\}$, and $P_i' = \max\{N_i, P_i\}$. Let $\tilde{C}_i = \langle C_i, N_i', P_i'\rangle$ and $\langle \tilde{C}_{i_1}, \ldots, \tilde{C}_{i_k}\rangle$ be the lexicographically sorted sequence of $\tilde{C}_i$'s, $S = |\{i \mid N_i' \neq N_i\}| \mod 2$. Then,

$$\mathsf{code}(f) = (\langle \tilde{C}_{i_1}, \ldots, \tilde{C}_{i_k}\rangle, 0, 0, S) \tag{3}$$

The following lemma describes some of the properties of the function code.

▶ **Lemma 11.** *(a) Let $f_1, \ldots, f_k$ be constant-free ROPs of Type-1, $a_1, \ldots, a_k, b_1, \ldots, b_k \in \{-1, 1\}$. Then*

$$\mathsf{code}(\prod_{i=1}^{k} a_i f_i) = \mathsf{code}(\prod_{i=1}^{k} b_i f_i) \iff \mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = \mathsf{parity}(\mathsf{sgn}(b_1), \ldots, \mathsf{sgn}(b_k)).$$

(b) $code(-\prod_{i=1}^{k} f_i) = (C, 0, 0, \bar{S})$, where $\mathsf{code}(\prod_{i=1}^{k} f_i) = (C, 0, 0, S)$.

(c) Let $f_1, \ldots, f_k$ be ROPs of Type-2 and suppose $\mathsf{code}(\sum_{i=1}^{k} f_i) = (C, N, P, 0)$. Then $\mathsf{code}(-\sum_{i=1}^{k} f_i) = (C, P, N, 0)$.

Proof. Proof is by induction on the number of variables in the constant-free read-once formula f. We consider two base cases. Let $f = \prod_{i=1}^{k} x_k$. Then by base case 4 in the definition of code,

$$\mathsf{code}(-f) = \left((\langle \emptyset, 1, 0\rangle, \ldots, \langle \emptyset, 0, 1\rangle), 0, 0, \bar{S}\right)$$

(a), (b) follow immediately now, and (c) is not relevant for this case. The second base case is when $f = \sum_i a_i x_i$. Note that only (c) is relevant here. Then $-f = \sum_i -a_i x_i$, and hence $\mathsf{code}(-f) = (C, P, N, 0)$, where $\mathsf{code}(f) = (C, N, P, 0)$. This proves (c) for the second base case.

Inductively suppose that statements (a)-(c) hold for all constant-free ROPs on $n' \leq n-1$ variables. Let f be a constant-free ROP of Type-2 on n variables, *i.e.* , $f = \prod_{i=1}^{k} f_i$, where f_is are constant-free ROPs of Type-1. Then, for $a = (a_1, \ldots, a_k) \in \{-1, 1\}^k$, $f_a = \prod_{i=1}^{k} a_i f_i$ is also a constant-free ROP of Type-2 on n variables. Suppose $\mathsf{code}(f_i) = (C_i, N_i, P_i, 0)$ for $1 \leq i \leq k$, then by (3),

$$\mathsf{code}(f) = ((\langle C_1, N_1', P_1'\rangle, \ldots, \langle C_1, N_k', P_k'\rangle), 0, 0, S).$$

By (c) of the induction hypothesis, we have $\mathsf{code}(-f_i) = (C_i, P_i, N_i, 0)$. Then, applying the construction given by (3),

$$\mathsf{code}(f_a) = (\mathsf{sort}(\langle C_1, N_1', P_1'\rangle, \ldots, \langle C_k, N_k', P_k'\rangle), 0, 0, S_a)$$

with $S_a = S$, if $\mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = 0$, and $S_a = \bar{S}$ otherwise. This proves (a) and (b).

To prove (c), suppose f is a constant-free ROP of Type-1 on n variables, i.e., $f = \sum_{i=1}^{k} f_i$. Suppose $\mathsf{code}(f_i) = (C_i, 0, 0, S_i)$, and $\langle C_1, \ldots, C_k\rangle$ be the lexicographically sorted order of C_i's, without loss of generality. Then, by the definition of code given in (2), $\mathsf{code}(f) = ((C_1, \ldots, C_k), N, P, 0)$, where $N = \mathsf{binary}(S_1, \ldots, S_k)$, and $P = \mathsf{binary}(\bar{S}_1, \ldots, \bar{S}_k)$. Applying induction hypothesis (b) on f_i, $\mathsf{code}(-f_i) = (C_i, 0, 0, \bar{S}_i)$. As $-f = \sum_{i=1}^{k} -f_i$, by (2) and the induction hypothesis, we have

$$\begin{aligned} \mathsf{code}(f) &= (\langle C_1, \ldots, C_k\rangle, \tilde{N}, \tilde{P}, 0) \text{ where} \\ \tilde{N} &= \mathsf{binary}(\bar{S}_1, \ldots, \bar{S}_k) \text{ and} \\ \tilde{P} &= \mathsf{binary}(S_1, \ldots, S_k) \end{aligned}$$

This implies $P = \tilde{N}$, and $N = \tilde{P}$, and hence (c) follows. ◀

Using these properties we prove that code is indeed a canonization for constant-free ROPs.

▶ **Lemma 12.** *Let f, and g be two constant-free ROPs. Then, $f \cong g \iff \mathsf{code}(f) = \mathsf{code}(g)$*

Proof. Proof is by induction on the structure and number of variables in f and g. For base Case, $f = \pm x_i, \sum_{i=1}^{k} a_i x_i$, or $\prod_{i=1}^{k} a_i x_i$, where $a_i \in \{-1, 1\}$. By examining the four base cases in the definition of code, the Lemma follows for these cases. For the induction step, we consider two cases depending on whether f is of Type-1 or Type-2.

Type 1: Let $f = f_1 + \ldots + f_k$ and $g = g_1 + \ldots + g_k$. First suppose $f \cong g$ via a bijection ϕ between the variables of f and g. As f_i's are variable disjoint, there exists a $\sigma \in \Sigma_k$ such that $\phi(f_i) = g_{\sigma(i)}$, and hence $f_i \cong g_{\sigma(i)}$, and by induction hypothesis, we have $\mathsf{code}(f_i) = \mathsf{code}(g_{\sigma(i)}) = (C_i, 0, 0, S_i)$. By (2), we can conclude that $\mathsf{code}(f) = \mathsf{code}(g)$. For the converse direction, suppose that $\mathsf{code}(f) = \mathsf{code}(g)$. Let $\mathsf{code}(f) = (\langle C_1, \ldots, C_k\rangle, \mathsf{binary}(S_1 \ldots S_k), \mathsf{binary}(\bar{S}_1, \ldots, \bar{S}_k), 0) = \mathsf{code}(g)$. Then by the structure of $\mathsf{code}(g)$ as in (2), we conclude $\mathsf{code}(f_i) = (C_i, 0, 0, S_i) = \mathsf{code}(g_i) \implies f_i \cong g_i$ (by induction hypothesis). Then, we have $g \cong f$ by Lemma 10.

Type 2: Let $f = f_1 \times f_2 \times \ldots f_k$ and $g = g_1 \times g_2 \times \ldots \times g_k$. Let $\mathsf{code}(f) = (C, 0, 0, S)$, and $\mathsf{code}(g) = (D, 0, 0, R)$, where $C = (\langle C_1, N_1', P_1'\rangle, \ldots, \langle C_k, N_k', P_k'\rangle)$, and $D = (\langle D_1, L_1', M_1'\rangle, \ldots, \langle D_k, L_k', M_k'\rangle)$. Suppose $\mathsf{code}(g) = \mathsf{code}(f)$. Then, by the definition of code, and Lemma 11, we have $\forall i \in [k]$, either $\mathsf{code}(f_i) = \mathsf{code}(g_i)$ or $\mathsf{code}(f_i) = \mathsf{code}(-g_i)$, and hence either $f_i \cong g_i$ or $f_i \cong -g_i$. As $S = R$, $|\{i \mid \mathsf{code}(f_i) = \mathsf{code}(-g_i)\}|$ must be even. Then by Lemma 10, we have $f \cong g$. For the converse direction, suppose $f \cong g$. Then by Lemma 10, there is a $\sigma \in \Sigma_k$, and $a_1, \ldots, a_k \in \{-1, 1\}$ with $\mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = 0$ such that $f_i \cong a_i g_{\sigma(i)}$, and hence $\mathsf{code}(f_i) = \mathsf{code}(a_i g_{\sigma(i)})$. Then, by the definition of code, we have $\mathsf{code}(\prod_{i=1}^{k} f_i) = \mathsf{code}(\prod_{i=1}^{k} a_i g_{\sigma(i)})$. As $\mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = 0$, by Lemma 11, $\mathsf{code}(\prod_{i=1}^{k} a_i g_{\sigma(i)}) = \mathsf{code}(\prod_{i=1}^{k} g_i)$, which completes the proof. ◀

▶ **Theorem 13.** *Isomorphism testing of constant-free read-once polynomials can be done in time polynomial in the number of variables in the input formulas.*

Proof. Given Lemma 12, the algorithm is obvious: on input f and g, compute $\mathsf{code}(f)$ and $\mathsf{code}(g)$, then check if $\mathsf{code}(f) = \mathsf{code}(g)$. Given f as an arithmetic constant-free read-once formula, $\mathsf{code}(f)$ can be computed in time polynomial in the size of the input formula. As size of $\mathsf{code}(.)$ as a collection of sets is at most the size of the input formula, we can test if $\mathsf{code}(f) = \mathsf{code}(g)$ in time linear in the size of the input formulas f and g. ◀

Extension to constant-free ROPs with arbitrary coefficients: The function code defined for constant-free ROPs can be extended to include constant-free ROPs where leaf nodes are labeled with $a_i x_i$, where $a_i \in \mathbb{Z}$. We denote this extension by general-constant-free ROPs. There is one main bottleneck for general-constant-free ROPs of Type-2: suppose $f = f_1 \times \cdots \times f_k$, and if a_1 is the GCD of coefficients of f_1, then $f = (f_1/a_1)(a_1 f_2) \times \cdots \times f_k$. So, a canonical code has to be invariant under taking out GCD of the coefficients of some of the f_i's and multiplying out these values among the remaining f_j's, provided the polynomial remains general-read-once. This can be achieved by explicitly carrying the GCD of the coefficients. For the sake of notational convenience, we denote the canonical function for general-constant-free ROPs by code'. For a general-constant-free ROP f, we define $\mathsf{code}'(f)$ as a quintuple (C, N, P, S, α), where C is a string, $N, P, \alpha \in \mathbb{N}$, $S \in \{0, 1\}$. The values C, P, N, and S have the same meaning as in the definition of code, and α is the GCD of the coefficients of f. We generalize the properties of code (details are skipped) to show:

▶ **Theorem 14.** *Isomorphism testing of general-constant-free ROPs can be done in* P.

Extension to Pre-processed ROPs: Motivated by [23], we extend Theorem 14 to the case of pre-processed constant-free ROPs. (See [23] for more on pre-processed ROPs). In a pre-processed constant-free ROP, a leaf labeled by variable x_i computes an arbitrary univariate polynomial $f_i(x_i)$ with coefficients from $\mathbb{Z}$. By providing an efficient way of canonically encoding the univariate polynomials that appear at the leaves, we obtain the following:

▶ **Theorem 15.** *Canonization of a pre-processed general constant-free ROP can be done in time polynomial in the number of variables and the degree of the univariate polynomials at the leaves.*

5 Polynomials with higher reads

As a natural extension, one could ask if the canonization procedure presented in the previous section can be extended to arithmetic formulas that read a variable at most twice. However, as in the case of Boolean formulas, it turns out that allowing variables twice makes the problem as hard as GI. In fact, even for the most primitive classes of Read-2 polynomials 1) Sum of two depth two monotone ROPs and, 2) Read-2 polynomials given in the $\Pi\Sigma$ form, isomorphism testing is complete for GI. It is already known that PI is harder than GI[24, 16]). We provide a different reduction that optimizes the number of reads. We skip some details here.

▶ **Theorem 16.** GI *polynomial time many-one reduces to testing isomorphism of two polynomials when both of the polynomials are given in one of the following representations:*

(a) Sum of two monotone read-once depth-2 arithmetic formulas with a + gate at the top.
(b) Read-2 monotone arithmetic formulas of depth two, with a × gate at the top.

Hardness of PI for the above special cases also forces one to ask whether the hardness given by Proposition 16 extends to the polynomial equivalence (PE) problem. Though we do not know the exact answer, we observe that PE for read-4 polynomials is hard for GI.

▶ Proposition 17. PE for the case of read-4 polynomials is hard for GI, for $\mathbb{F} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$.

References

1 M. Agrawal and N. Saxena. Equivalence of f-algebras and cubic forms. In *STACS*, pages 115–126, 2006.
2 M. Agrawal and T. Thierauf. The Formula Isomorphism Problem. *SIAM J. Comput.*, 30(3):990–1009, 2000.
3 A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesly, 1974.
4 D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, January 1993.
5 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
6 V. Arvind and J. Torán. Isomorphism testing: Perspective and open problems. *Bulletin of the EATCS*, 86:66–84, 2005.
7 B. Borchert, D. Ranjan, and F. Stephan. On the complexity of some classical equivalence relations on boolean functions. *Theory of Computing Systems*, 31(6):679–693, 1998.
8 D. Bshouty and N. H. Bshouty. On learning arithmetic read-once formulas with exponentiation (extended abstract). In *COLT*, pages 311–317, 1994.
9 N. H. Bshouty and R. Cleve. Interpolating arithmetic read-once formulas in parallel. *SIAM J. Comput.*, 27(2):401–413, 1998.
10 N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning arithmetic read-once formulas. *SIAM J. Comput.*, 24(4):706–735, 1995.
11 P. Clote and E. Kranakis. Boolean functions, invariance groups, and parallel complexity. *SIAM J. Comput.*, 20(3):553–590, 1991.
12 S. A. Cook and P. McKenzie. Problems complete for L. *Jl. of Algorithms*, 8:385–394, 1987.
13 J. Goldsmith, M. Hagen, and M. Mundhenk. Complexity of dnf minimization and isomorphism testing for monotone formulas. *Inf. Comput.*, 206(6):760–775, 2008.
14 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.
15 N. Kayal. Affine projections of polynomials. ECCC-Report TR11-061, 2011.
16 N. Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *SODA*. SIAM, 2011.
17 J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its structural complexity*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1993.
18 S. Lindell. A logspace algorithm for tree canonization (extended abstract). In *STOC*, pages 400–404, 1992.
19 J. Patarin. Hidden fields equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In *EUROCRYPT'96*, pages 33–48, 1996.
20 B. V. R. Rao and J. M. N. Sarma. On the complexity of matroid isomorphism problems. In *CSR*, pages 286–298, 2009.
21 N. Saxena. *Morphisms of Rings and Applications to Complexity*. PhD thesis, Department of Computer Science, Indian Institute of Technology, Kanpur, India, 2006.
22 A. Shpilka and I. Volkovich. Read-once polynomial identity testing. In *STOC*, pages 507–516, 2008.
23 A. Shpilka and I. Volkovich. Improved polynomial identity testing for read-once formulas. In *APPROX-RANDOM*, pages 700–713, 2009.
24 T. Thierauf. The isomorphism problem for read-once branching programs and arithmetic circuits. *Chicago J. Theor. Comput. Sci.*, 1998, 1998.

The Limited Power of Powering: Polynomial Identity Testing and a Depth-four Lower Bound for the Permanent

Bruno Grenet[1], Pascal Koiran[1], Natacha Portier*[1], and Yann Strozecki[2]

1 **LIP, UMR 5668, ÉNS de Lyon – CNRS – UCBL – INRIA**
École Normale Supérieure de Lyon, Université de Lyon
[Bruno.Grenet,Pascal.Koiran,Natacha.Portier]@ens-lyon.fr
2 **Équipe de Logique Mathématique, Université Paris VII**
Strozecki@logique.jussieu.fr

Abstract

Polynomial identity testing and arithmetic circuit lower bounds are two central questions in algebraic complexity theory. It is an intriguing fact that these questions are actually related. One of the authors of the present paper has recently proposed a "real τ-conjecture" which is inspired by this connection. The real τ-conjecture states that the number of real roots of a sum of products of sparse univariate polynomials should be polynomially bounded. It implies a superpolynomial lower bound on the size of arithmetic circuits computing the permanent polynomial.

In this paper we show that the real τ-conjecture holds true for a restricted class of sums of products of sparse polynomials. This result yields lower bounds for a restricted class of depth-4 circuits: we show that polynomial size circuits from this class cannot compute the permanent, and we also give a deterministic polynomial identity testing algorithm for the same class of circuits.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, I.1 Symbolic and Algebraic Manipulation

Keywords and phrases Algebraic Complexity, Sparse Polynomials, Descartes' Rule of Signs, Lower Bound for the Permanent, Polynomial Identity Testing

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.127

1 Introduction

The τ-conjecture [18, 19] states that a univariate polynomial with integer coefficients defined by an arithmetic circuit has a number of integer roots polynomial in the size of the circuit. A real version of this conjecture was recently presented in [14]. The real τ-conjecture states that the number of real roots of a sum of products of sparse univariate polynomials should be polynomially bounded as a function of the size of the corresponding expression. More precisely, consider a polynomial of the form

$$f(X) = \sum_{i=1}^{k} \prod_{j=1}^{m} f_{ij}(X),$$

* This material is based on work supported in part by the European Community under contract PIOF-GA-2009-236197 of the 7th PCRD.
This work was done while the authors were visiting the University of Toronto.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 127–139

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where $f_{ij} \in \mathbb{R}[X]$ has at most t monomials. The conjecture asserts that the number of real roots of f is bounded by a polynomial function of kmt. It was shown in [14] that this conjecture implies a superpolynomial lower bound on the arithmetic circuit complexity of the permanent polynomial (a central goal of algebraic complexity theory ever since Valiant's seminal work [20]). In this paper we show that the conjecture holds true in a special case. We focus on the case where the number of distinct sparse polynomials is small (but each polynomial may be repeated many times). We therefore consider expressions of the form

$$\sum_{i=1}^{k} \prod_{j=1}^{m} f_j^{\alpha_{ij}}(X). \tag{1}$$

We obtain a $O(t^{m(2^{k-1}-1)})$ upper bound on the number of real roots of such a polynomial, where t is the maximum number of monomials in the f_j's. In particular, the bound is polynomial in t when the "top fan-in" k and the number m of sparse polynomials in the expression are both constant. Note also that the bound is independent of the magnitude of the integers α_{ij}.

From this upper bound we obtain a lower bound on the complexity of the permanent for a restricted class of arithmetic circuits. The circuits that we consider are again of form (1), but now X should be interpreted as the tuple of inputs to the circuit rather than as a single real variable. Roughly speaking, we show a superpolynomial lower bound on the complexity of the permanent in the case where k and m are again fixed. More precisely, we show that a circuit of form (1) cannot compute the permanent as long as the sparsity of the f_j's is polynomially bounded. Note that this is a lower bound for a restricted class of depth-4 circuits: The output gate at depth 4 has fan-in bounded by the constant k, and the gates at depth 2 are only allowed to compute a constant (m) number of distinct polynomials f_j.

Our third main result is a deterministic identity testing algorithm, again for polynomials of the same form. When k and m are fixed, we can test if the polynomial in (1) is identically equal to 0 in time polynomial in t and in $\max_{ij} \alpha_{ij}$. Note that if k, m *and the exponents* α_{ij} are all bounded by a constant then the number of monomials in such a polynomial is $t^{O(1)}$ and our three main results become trivial. These results are therefore interesting only in the case where the α_{ij} may be large, and can be interpreted as limits on the power of powering.

1.1 Connection to Previous Work

The idea of deriving lower bounds on arithmetic circuit complexity from upper bounds on the number of real roots goes back at least to a 1976 paper by Borodin and Cook [6]. Their results were independently improved by Grigoriev and Risler (see [8], chapter 12). For a long time, it seemed that the lower bounds that can be obtained by this method had to be rather small since the number of real roots of a polynomial can be exponential in its arithmetic circuit size. Nevertheless, as explained above it was recently shown in [14] that superpolynomial lower bounds on the complexity of the permanent on general arithmetic circuits can be derived from a suitable upper bound on the number of roots of sums of products of sparse polynomials. This is related to the fact that for low degree polynomials, arithmetic circuits of depth 4 are almost equivalent to general arithmetic circuits [3, 13].

The study of polynomial identity testing (PIT) also has a long history. The Schwartz-Zippel lemma [17] yields a randomized algorithm for PIT. A connection between deterministic PIT and arithmetic circuit lower bounds was pointed out as early as 1980 by Heintz and Schnorr [11], but a more in-depth study of this connection began only much later [12]. The recent literature contains deterministic PIT algorithms for various restricted models (see

e.g. the two surveys [2, 16]). These algorithms are either black-box, i.e. the algorithm can only test the circuit for zero on inputs of its own choosing, or non-black-box in which case the algorithm has access to the structure of the circuit. One model which is similar to ours was recently studied in [5]. It follows from Theorem 1 in [5] that there is a polynomial time deterministic black-box PIT algorithm for polynomials of the form (1) if, instead of bounding k and m as in our algorithm, we bound the transcendence degree r of the polynomials f_j. Obviously we have $r \leq m$, so from this point of view their result is more general.[1] On the other hand their running time is polynomial in the degree of the f_j, whereas we can handle polynomials of exponential degree in polynomial time. Furthermore they not provide any lower bound result for the permanent. (Note that the bound that is deduced from their black-box PIT algorithm using the technique of [1] only applies to polynomial families with coefficients in PSPACE and not to the permanent family.)

1.2 Our approach

The proof of our bound on the number of real roots has the same high-level structure as that of Descartes' rule of signs.

▶ **Proposition 1.** *A univariate polynomial $f \in \mathbb{R}[X]$ with $t \geq 1$ monomials has at most $t-1$ positive real roots.*

The number of negative roots of f is also bounded by $t-1$ (consider $f(-X)$), hence there are at most $2t-1$ real roots (including 0). There is also a refined version of Proposition 1 where the number of monomials t is replaced by the number of sign changes in the sequence of coefficients of f. The cruder version will be sufficient for our purposes.

We briefly recall an inductive proof of Proposition 1. For $t = 1$, there is no non-zero root. For $t > 1$, let $a_\alpha X^\alpha$ be the monomial of lowest degree. We can assume that $\alpha = 0$ (if not, we can divide f by X^α since this operation does not change the number of positive roots). Consider now the derivative f'. It has $t-1$ monomials, and at most $t-2$ positive real roots by induction hypothesis. Moreover, by Rolle's theorem there is a positive root of f' between 2 consecutive positive roots of f. We conclude that f has at most $(t-2)+1 = t-1$ positive roots.

In (1) we have a sum of k terms instead of t monomials, but the basic strategy remains the same: we divide by the first term and take the derivative. This has the effect of removing a term, but it also has the effect (unlike Descartes' rule) of increasing the complexity of the remaining $k-1$ terms. This results in a larger bound (and a longer proof).

From this upper bound we obtain our permanent lower bound by applying the proof method which was put forward in [14]. More precisely, assume that the permanent has an efficient representation of the form (1). We show that the same must be true for the univariate polynomial $\prod_{i=1}^{2^n}(X-i)$ using a result of Bürgisser [7]. This yields a contradiction with our upper bound on the number of real roots.

Our third result is a polynomial identity testing algorithm. Using a standard substitution technique, we can assume that the polynomials f_j in (1) are univariate. We note that the resulting f_j may be of exponential degree even if the original multivariate f_j are of low degree. The construction of hitting sets is a classical approach to deterministic identity testing. Recall that a hitting set for a class $\mathcal{F}$ of polynomials is a set of points H such that for any non-identically zero polynomial $f \in \mathcal{F}$ we have a point $x \in H$ such that $f(x) \neq 0$.

[1] As pointed out by the authors of [5], their result already seems nontrivial for a constant m.

Clearly, a hitting set yields a black-box identity testing algorithm (it is not hard to see that the converse is also true). Moreover, for any class $\mathcal{F}$ of univariate polynomials, an upper bound $z(\mathcal{F})$ on the number of real roots of each non-zero polynomial in $\mathcal{F}$ yields a hitting set (any set of $z(\mathcal{F})+1$ real numbers will do). From our upper bound result we therefore have polynomial size hitting sets for polynomials of the form (1) when k and m are fixed. Unfortunately, the resulting black-box algorithm does not run in polynomial time: evaluating a polynomial at a point of the hitting set may not be feasible in polynomial time since (as explained above) the f_j may be of very high degree. We therefore use a different strategy. Roughly speaking, we "run" the proof of our upper bound theorem on an input of form (1). This requires explicit knowledge of this representation, and the resulting algorithm is non-black-box. As explained in Section 1.1, for the case where the f_j are low-degree multivariate polynomials an efficient black-box algorithm was recently given in [5].

Organization of the paper. In Section 2 we prove an upper bound on the number of real roots of polynomials of the form (1), see Theorems 10 and 11 at the end of the section. In fact, we obtain an upper bound for a more general class of polynomials which we call $\mathsf{SPS}(k,m,t,h)$. This generalization is needed for the inductive proof to go through. From this upper bound, we derive in Section 3 a lower bound on the computational power of (multivariate) circuits of the same form. We give in Section 4 a deterministic identity testing algorithm, again for polynomials of form (1).

2 The real roots of a sum of products of sparse polynomials

2.1 Definitions

In this section, we define precisely the polynomials we are working with. We then explain how to transform those polynomials in a way which reduces the number of terms but does not increase too much the number of roots. This method has some similarities with the proof of Lemma 2 in [15] and it leads to a bound on the number of roots of the polynomials we study.

We say that a polynomial is *t-sparse* if it has at most t monomials.

▶ **Definition 2.** Let $\mathsf{SPS}(k,m,t,h)$ denote the class of polynomials $\phi \in \mathbb{R}[X]$ defined by

$$\phi(X) = \sum_{i=1}^{k} g_i(X) \prod_{j=1}^{m} f_j^{\alpha_{ij}}(X)$$

where

- $g_1, \ldots, g_k$ are h-sparse polynomials over $\mathbb{R}$;
- $f_1, \ldots, f_m$ are t-sparse non-zero polynomials over $\mathbb{R}$;
- $\alpha_{11}, \ldots, \alpha_{km}$ are non-negative integers.

We define $P_i = \prod_{j=1}^{m} f_j^{\alpha_{ij}}$ and $T_i = g_i P_i$ for all i. We also define $\pi = \prod_{j=1}^{m} f_j$. Finally, we define $\mathsf{SPS}(k,m,t)$ as the subclass of $\mathsf{SPS}(k,m,t,h)$ in which all the g_i are equal to the constant 1.

Note that $\mathsf{SPS}(k,m,t)$ is just the class of polynomials of form (1), and is included in $\mathsf{SPS}(k,m,t,1)$. We want to give a bound for the number of real roots of the polynomials in this class, and more generally in $\mathsf{SPS}(k,m,t,h)$. To this end, from a polynomial $\phi \in \mathsf{SPS}(k,m,t,h)$, we build in Lemma 4 a new polynomial $\tilde{\phi} \in \mathsf{SPS}(k-1,m,t,\tilde{h})$ for some $\tilde{h}$ such that a bound on the number of real roots of $\tilde{\phi}$ yields a bound for ϕ. We first give a

bound for the number of roots of the polynomials in $\mathsf{SPS}(2, m, t)$. The proof in this case contains the main ingredients of the general case with less technicalities.

▶ **Proposition 3.** *Let $\phi = \prod_{j=1}^m f_j^{\alpha_{1j}} + \prod_{j=1}^m f_j^{\alpha_{2j}} \in \mathsf{SPS}(2, m, t)$. Then ϕ has at most kmt real roots.*

Proof. Let $\psi = \phi / \prod_j f_j^{\alpha_{1j}} = 1 + \prod_{j=1}^m f_j^{\alpha_{2j}-\alpha_{1j}}$. Then

$$\psi' = \prod_{j=1}^m f_j^{\alpha_{2j}-\alpha_{1j}-1} \times \sum_{j=1}^m (\alpha_{2j} - \alpha_{1j}) f_j' \prod_{l \neq j} f_l.$$

Since each f_j is t-sparse, the polynomial $\sum_{j=1}^m (\alpha_{2j} - \alpha_{1j}) f_j' \prod_{l \neq j} f_l$ has at most mt^m monomials. Therefore, its number of real roots is at most $2mt^m - 1$ (by Descartes' rule of signs). The number of roots and poles of the rational function $\prod_{j=1}^m f_j^{\alpha_{2j}-\alpha_{1j}-1}$ is at most $2m(t-1)$, the total number of roots of the f_j's. Therefore, the number of roots and poles of ψ' is at most $2mt^m - 1 + 2m(t-1)$. Now, between two consecutive roots of the rational function ψ, there exists a root or a pole of ψ', so there ψ has at most $2m(t^m + t - 1)$ roots. Since the number of roots of ϕ is bounded by the sum of the number of roots of ψ and $\prod_j f_j^{\alpha_{1j}}$, ϕ has at most $2m(t^m + t - 1) + 2m(t-1)$ real roots. ◀

We now turn to the general case.

▶ **Lemma 4.** *Let $\phi \in \mathsf{SPS}(k, m, t, h)$. If g_1 is not identically zero, let $\psi = \phi/T_1$ and $\tilde{\phi} = g_1 T_1 \pi \psi'$; otherwise let $\tilde{\phi} = \phi$. Then there exists $\tilde{h}$ such that $\tilde{\phi} \in \mathsf{SPS}(k-1, m, t, \tilde{h})$.*

Proof. If g_1 is identically zero, the theorem holds with $\tilde{h} = h$. Assume now that g_1 is not identically zero. Then

$$\psi(X) = \phi(X)/T_1(X) = 1 + \frac{1}{T_1(X)} \cdot \sum_{i=2}^k T_i(X)$$

and

$$\psi' = \frac{\sum_{i=2}^k (T_1 T_i' - T_1' T_i)}{T_1^2}.$$

Notice that $T_i' = g_i' P_i + g_i P_i'$ and

$$P_i' = \sum_{j=1}^m \alpha_{ij} f_j' f_j^{\alpha_{ij}-1} \cdot \prod_{l \neq j} f_l^{\alpha_{il}} = P_i \cdot \sum_{j=1}^m \alpha_{ij} f_j' / f_j.$$

Therefore

$$\begin{aligned}
\psi' &= \frac{1}{T_1^2} \cdot \sum_{i=2}^k (g_1 P_1 g_i' P_i + g_1 P_1 g_i P_i' - g_1' P_1 g_i P_i - g_1 P_1' g_i P_i) \\
&= \frac{1}{T_1^2} \cdot \sum_{i=2}^k (g_1 g_i' P_1 P_i + g_1 g_i P_1 P_i \sum_j \alpha_{ij} f_j'/f_j \\
&\qquad\qquad - g_1' g_i P_1 P_i - g_1 g_i P_1 P_i \sum_j \alpha_{1j} f_j'/f_j) \\
&= \frac{1}{g_1 T_1} \cdot \sum_{i=2}^k P_i \left(g_1 g_i' - g_1' g_i + g_1 g_i \sum_j (\alpha_{ij} - \alpha_{1j}) f_j'/f_j \right).
\end{aligned}$$

We now multiply ψ' by $\pi = \prod_j f_j$ and get

$$\pi\psi' = \frac{1}{g_1 T_1} \cdot \sum_{i=2}^{k} P_i \left(\pi \cdot (g_1 g_i' - g_1' g_i) + g_1 g_i \sum_j (\alpha_{ij} - \alpha_{1j}) f_j' \prod_{l \neq j} f_l \right).$$

Thus $g_1 T_1 \pi \psi'$ is a polynomial of the class $\mathsf{SPS}(k-1, m, t, \tilde{h})$ for some $\tilde{h}$. Let us write

$$\tilde{\phi} = g_1 T_1 \pi \psi' = \sum_{i=2}^{k} P_i \tilde{g}_i.$$

The integer $\tilde{h}$ denotes the maximum number of monomials in $\tilde{g}_i$ for $2 \leq i \leq k$. ◀

▶ **Definition 5.** Let $(\phi_n)_{1 \leq n \leq k}$ be the sequence defined by $\phi_1 = \phi$ and for $n \geq 1$, $\phi_{n+1} = \tilde{\phi}_n$. Let also, for $1 \leq i \leq k$, $(g_i^{(n)})_{1 \leq n \leq i}$ be such that for $1 \leq n \leq k$,

$$\phi_n = \sum_{i=n}^{k} g_i^{(n)} \prod_{j=1}^{m} f_j^{\alpha_{ij}}.$$

We also define the sequence $(h_n)_{1 \leq n \leq k}$ by $h_1 = 1$ and $h_{n+1} = \tilde{h}_n$. That is, each $g_i^{(n)}$ is h_n-sparse.

2.2 A generalization of Descartes' rule

In Definition 5 we defined a sequence of polynomials (ϕ_n) and a sequence of integers (h_n). In this section we first prove that the number of real roots of ϕ_n is bounded by the number of real roots of ϕ_{n+1} up to a multiplicative constant. Then, we give an upper bound on h_n and we combine these ingredients to obtain a bound on the number of real roots of a polynomial in $\mathsf{SPS}(k, m, t)$. This bound (in Theorem 10 at the end of the section) is polynomial in t.

We denote by $r(P)$ the number of distinct real roots of a rational function P. In order to obtain a bound on $r(\phi)$ from a bound on $r(\tilde{\phi})$, we need the following lemma.

▶ **Lemma 6.** *Let $P \in \mathsf{SPS}(1, m, t, h)$. If P is not identically zero then*

$$r(P) \leq 2h + 2m(t-1) - 1.$$

Proof. By definition, $P = g \cdot \prod_j f_j^{\alpha_j}$. The number of non-zero real roots of P is therefore bounded by the sum of the number of non-zero real roots of g and of the f_j's. Since g is h-sparse, we know from Descartes' rule that is has at most $2(h-1)$ non-zero real roots. Likewise, each f_j has at most $2(t-1)$ real roots. As a result, P has at most $2(h-1)+2m(t-1)$ non-zero real roots. Since 0 can also be a root, we add 1 to this bound to obtain the final result. ◀

▶ **Lemma 7.** *Let $\phi \in \mathsf{SPS}(k, m, t, h)$. Then*

$$r(\phi) \leq r(\tilde{\phi}) + 4h + 4m(t-1) - 1.$$

Proof. If g_1 is zero in the definition of ϕ, then $\tilde{\phi} = \phi$ which proves the lemma.

Recall from the proof of Lemma 4 the notation $\psi = \phi / T_1$. If g_1 is not identically zero, by definition we have $\tilde{\phi} = g_1 T_1 \pi \psi'$, so the number $r(\tilde{\phi})$ of real roots of the polynomial $\tilde{\phi}$ is an upper bound on the number of real roots of ψ'.

Since $\phi = T_1\psi$, we have $r(\phi) \leq r(T_1) + r(\psi)$. Moreover, between two consecutive roots of the rational function ψ, we have a root of ψ' or a root of the denominator T_1. As a result, $r(\psi) \leq r(\psi') + r(T_1) + 1$. It follows that $r(\phi) \leq r(\psi') + 2r(T_1) + 1 \leq r(\tilde{\phi}) + 2r(T_1) + 1$. Moreover, the polynomial $T_1 = g_1 \cdot \prod_j f_j^{\alpha_{1j}}$ is in $\mathsf{SPS}(1, m, t, h)$. Thus by Lemma 6, T_1 has at most $2h + 2m(t-1) - 1$ real roots. We conclude that ϕ has at most

$$r(\tilde{\phi}) + 2 \cdot (2h + 2m(t-1) - 1) + 1 = r(\tilde{\phi}) + 4h + 4m(t-1) - 1$$

real roots. ◀

▶ **Proposition 8.** *Let $\phi \in \mathsf{SPS}(k, m, t, 1)$. Then*

$$r(\phi) \leq 2h_k + 4\sum_{i=1}^{k-1} h_i + 2m(2k-1)(t-1) - k.$$

Proof. Lemma 7 gives the following recurrence:

$$r(\phi_n) \leq r(\phi_{n+1}) + 4h_n + 4m(t-1) - 1.$$

Thus, we get

$$r(\phi) \leq r(\phi_k) + 4\sum_{i=1}^{k-1} h_i + (k-1)(4m(t-1) - 1). \tag{2}$$

Since $\phi_k \in \mathsf{SPS}(1, m, t, h_k)$, Lemma 6 bounds its number of real roots:

$$r(\phi_k) \leq 2h_k + 2m(t-1) - 1. \tag{3}$$

The bound is a combination of (2) and (3). ◀

Proposition 8 shows that in order to bound $r(\phi)$, we need a bound on h_n.

▶ **Proposition 9.** *For all n, h_n is bounded by $((m+2)t^m)^{2^{n-1}-1}$.*

Proof. As showed in the proof of Lemma 4, $\tilde{\phi} = \sum_{i=2}^{k} \tilde{g}_i P_i$ where each $\tilde{g}_i$ is $\tilde{h}$-sparse. More precisely,

$$\tilde{g}_i = (g_1 g_i' - g_1' g_i) \prod_{j=1}^{m} f_j + g_1 g_i \sum_{j=1}^{m} (\alpha_{ij} - \alpha_{1j}) f_j' \prod_{l \neq j} f_l.$$

Thus $\tilde{g}_i$ is a sum of $(m+2)$ terms, and each term is a product of m t-sparse polynomials by two h-sparse polynomials. Thus $\tilde{h} \leq (m+2)t^m h^2$.

This gives the following recurrence relation on h_n:

$$\begin{cases} h_1 & = 1 \\ h_{n+1} & \leq (m+2)t^m h_n^2 \end{cases}$$

Therefore, $h_n \leq ((m+2)t^m)^{2^{n-1}-1}$. ◀

Now, we combine Propositions 8 and 9 to obtain our first bound on the number of roots of a polynomial in $\mathsf{SPS}(k, m, t)$.

▶ **Theorem 10.** *Let $\phi \in \mathsf{SPS}(k, m, t)$: we have $\phi = \sum_{i=1}^{k} \prod_{j=1}^{m} f_j^{\alpha_{ij}}$ where for all i and j, f_j is t-sparse and $\alpha_{ij} \geq 0$. Then $r(\phi) \leq C \times ((m+2)t^m)^{2^{k-1}-1}$ for some universal constant C.*

The bound for $\tilde{h}$, the number of monomials in the polynomials $g_i^{(n)}$, can actually be improved. This automatically sharpens the bound we give for the number of real roots of a polynomial in $\mathsf{SPS}(k, m, t)$.

▶ **Theorem 11.** *Let $\phi \in \mathsf{SPS}(k, m, t)$. Then ϕ has at most*

$$C \times \left[e \times \left(1 + \frac{t^m}{2^{k-1} - 1} \right) \right]^{2^{k-1}-1}$$

real roots, where C is a universal constant.

The proofs of these two theorems can be found in the full version of this paper [10, Theorem 1 and 2].

3 Lower bounds

In this section we introduce a subclass $\mathsf{mSPS}(k, m)$ of the class of "easy to compute" multivariate polynomial families, and we use the results of Section 2.2 to show that it does not contain the permanent family. The polynomials in a $\mathsf{mSPS}(k, m)$ family have the same structure as the univariate polynomials in the class $\mathsf{SPS}(k, m, t)$ from Definition 2. In this section, polynomial families are denoted by their general term in brackets: The polynomial P_n is the n-th polynomial of the family (P_n). When there is no ambiguity on the number of variables, we denote by $\vec{X}$ the tuple of variables of a polynomial P_n. The definition uses the notion of *constant-free* circuit: An arithmetic circuit is said constant-free if the only constant input is -1 (or equivalently are of polynomially bounded bitsize).

▶ **Definition 12.** We say that a sequence of polynomials (P_n) is in $\mathsf{mSPS}(k, m)$ if there is a polynomial Q such that for all n:

(i) P_n depends on at most $Q(n)$ variables.
(ii) $P_n(\vec{X}) = \sum_{i=1}^{k} \prod_{j=1}^{m} f_{jn}^{\alpha_{ij}}(\vec{X})$
(iii) The bitsize of α_{ij} is bounded by $Q(n)$.
(iv) For all $1 \leq j \leq m$, the polynomial f_{jn} has a constant-free circuit of size $Q(n)$ and is $Q(n)$-sparse.

▶ Remark. If $(P_n) \in \mathsf{mSPS}(k, m)$ then each P_n has a constant-free circuit of size polynomial in n. Indeed from the constant-free circuits of the polynomials f_{jn} we can build a constant-free circuit for P_n. We have to take the α_{ij}-th power of f_{jn}, which can be done with a circuit of size polynomial in the bitsize of α_{ij} thanks to fast exponentiation. The size of the final circuit is up to a constant the sum of the sizes of these powering circuits and of the circuits giving f_{jn}, which is thus polynomial in n.

▶ **Definition 13.** The Pochhammer-Wilkinson polynomial of order 2^n is defined by $\mathrm{PW}_n = \prod_{i=1}^{2^n} (X - i)$.

▶ **Definition 14.** The Permanent over n^2 variables is defined by $\mathrm{PER}_n = \sum_{\sigma \in \Sigma_n} \prod_{i=1}^{n} X_{i\sigma(i)}$ where Σ_n is the set of permutations of $\{1, \ldots, n\}$.

We now give a lower bound on the Permanent, using its completeness for VNP [20], a result of Bürgisser on the Pochhammer-Wilkinson polynomials [7] and our bound on the roots of the polynomials in $\mathsf{SPS}(k, m, t)$. We refer to Bürgisser's book [9] for the definition and properties of VNP.

▶ **Theorem 15.** *The family of polynomials* (PER_n) *is not in* $\mathsf{mSPS}(k, m)$ *for any fixed* k *and* m*, i.e., there is no representation of the permanent family of the form*

$$\mathrm{PER}_n(\vec{X}) = \sum_{i=1}^{k} \prod_{j=1}^{m} f_{jn}^{\alpha_{ij}}(\vec{X})$$

where k *and* m *are constant and the bitsize of the* α_{ij}*, the sparsity of the polynomials* f_{jn} *and their constant-free arithmetic circuit complexity are all bounded by a polynomial function* $Q(n)$*.*

Proof. Assume by contradiction that $(\mathrm{PER}_n) \in \mathsf{mSPS}(k, m)$. By the previous remark, this implies that PER_n can be computed by polynomial size constant-free arithmetic circuits. As in the proofs of Theorem 4.1 and 1.2 in [7], it follows from this property that there is a family $(G_n(X_0, \ldots, X_n))$ in VNP such that

$$\mathrm{PW}_n(X) = G_n(X^{2^0}, X^{2^1}, \ldots, X^{2^n}). \tag{4}$$

Since the permanent is complete for VNP, we have a polynomial h such that

$$\mathrm{PER}_{h(n)}(z_1, \ldots, z_{h(n)^2}) = G_n(X_0, \ldots, X_n) \tag{5}$$

where the z_i's are either variables of G_n or constants. By hypothesis $(\mathrm{PER}_n) \in \mathsf{mSPS}(k, m)$. Let Q be the corresponding polynomial from Definition 12. From this definition and from (4) and (5) we have

$$\mathrm{PW}_n(X) = \sum_{i=1}^{k} \prod_{j=1}^{m} f_{jn}(X)^{\alpha_{ij}}$$

where $f_{jn}(X)$ is $Q(h(n))$-sparse. This shows that the polynomial PW_n is in $\mathsf{SPS}(k, m, R(n))$ where $R(n) = Q(h(n))$.

We have proved in Theorem 10 that polynomials in $\mathsf{SPS}(k, m, R(n))$ have at most $r(n) = C \times ((m+2)R(n))^{m})^{2^{k-1}-1}$ real roots. On the other hand, by construction the polynomial PW_n has 2^n roots, which is larger than $r(n)$ for all large enough n. This yields a contradiction and completes the proof of the theorem. ◀

Theorem 15 gives a lower bound for a restricted class of depth-4 circuits: The top fan-in is bounded by k, and the gates at depth 2 compute only m distinct polynomials f_j. Yet, each f_j can be duplicated an exponential number of times so that the gates at depth 3 have an unbounded fan-in. Therefore, the lower bound holds for a class of exponential-size depth-4 circuits. Note that the result is already non trivial for polynomial-size depth-4 circuits of this kind.

▶ Remark. It is possible to relax condition (iv) in Definition 12. We can replace it by the less restrictive condition:

(iv') *the polynomial* f_{jn} *is* $Q(n)$*-sparse,*

i.e., we allow polynomials f_{jn} with arbitrary complex coefficients. Theorem 15 still applies to this larger version of the class $\mathsf{mSPS}(k,m)$, but for the proof to go through we need to assume the Generalized Riemann Hypothesis. The only change is at the beginning of the proof: Assuming that the permanent family belongs to the (redefined) class $\mathsf{mSPS}(k,m)$, we can conclude that this family can be computed by polynomial size arithmetic circuits with arbitrary constants. To see this, note that any non-multilinear monomial in any f_{jn} can be deleted since it cannot contribute to the final result (the permanent is multilinear). And since f_{jn} is sparse, there is a polynomial size arithmetic circuit with arbitrary constants to compute its multilinear monomials. The remainder of the proof is essentially unchanged. But to deal with arithmetic circuits with arbitrary constants (from the complex field) instead of constant-free arithmetic circuits, we shall use Corollary 4.2 of [7] instead of Theorems 1.2 and 4.1. This means that we have to assume GRH as in this corollary. It is an intriguing question whether this assumption can be removed from Corollary 4.2 of [7] and from this lower bound result.

4 Polynomial Identity Testing

This section is devoted to a proof that Identity Testing can be done in deterministic polynomial time on the polynomials studied in the previous sections. Recall from Definition 5 that for $\phi = \sum_{i=1}^{k} P_i \in \mathsf{SPS}(k,m,t)$, (ϕ_n) is defined by $\phi_n = \sum_{i=n}^{k} g_i^{(n)} P_i$.

▶ **Lemma 16.** *Let $\phi \in \mathsf{SPS}(k,m,t)$ and (ϕ_n) as in Definition 5. Then for $l < k$, $\phi_l \equiv 0$ if and only if $\phi_{l+1} \equiv 0$ and ϕ_l has a smaller degree than $g_l^{(l)} P_l$.*

Proof. If for all i, $g_i^{(l)}$ is identically zero, then the lemma holds. If there is at least one which is not identically zero, assume that it is $g_l^{(l)}$ up to a reindexing of the terms.

Let $T_l = g_l^{(l)} P_l$, recall that $\phi_{l+1} = g_l T_l \pi(\phi_l/T_l)'$. If $\phi_l \equiv 0$, then $\phi_{l+1} \equiv 0$. Moreover, we have assumed that $T_l \not\equiv 0$ and it is thus of larger degree than ϕ_l which is identically 0.

Assume now that $\phi_{l+1} \equiv 0$, that is $g_l T_l \pi(\phi_l/T_l)' \equiv 0$. By hypothesis, T_l and π are not identically zero, therefore $(\phi_l/T_l)' \equiv 0$. Thus there is $\lambda \in \mathbb{R}$ such that $\phi_l = \lambda T_l$. Since by hypothesis ϕ_l and T_l have different degrees, $\lambda = 0$ and $\phi_l \equiv 0$. ◀

To solve PIT, we will need to explicitly compute the sequence of polynomials ϕ_l. Thus, the algorithm is not black-box: it must have access to a representation of the input polynomial under form (1).

▶ **Theorem 17.** *Let k and m be two integers and $\phi \in \mathsf{SPS}(k,m,t)$: we have $\phi = \sum_{i=1}^{k} \prod_{j=1}^{m} f_j^{\alpha_{ij}}$ where for all i and j, f_j is t-sparse and $\alpha_{ij} \geq 0$. Then one can test if ϕ is identically zero in time polynomial in t, in the size of the sparse representation of the f_j's and in the α_{ij}'s.*

Proof. Let (ϕ_n) be the sequence defined from ϕ as in Definition 5. Lemma 16 implies that ϕ is identically zero if and only if ϕ_k is identically zero and that for all $l < k$, $\phi_l = \sum_{i=l}^{k} g_i^{(l)} P_i$ has a strictly smaller degree than $g_l^{(l)} P_l$.

First, one computes the sparse polynomials $g_i^{(l)}$ for all i and l as sums of monomials. It is done in time polynomial in the size of the f_j's since k and m are fixed. We can then verify if ϕ_k is identically zero.

We now want to test for each l if the degrees of $g_l^{(l)} P_l$ and of ϕ_l differ. First remark that we know the highest degree monomial of $g_i^{(l)}$ for $i \geq l$ since we have computed all the $g_i^{(l)}$'s. One can also compute the highest degree monomial of each P_i in time polynomial in the α_{ij}'s (not their bitsize) and the size of the f_j's. We have thus computed the degree

of $g_i^{(l)} P_i$ for all i and l and we reorder them so that $g_l^{(l)} P_l$ is of highest degree amongst them. Let S denote the sum of the highest degree monomial of $g_i^{(l)} P_i$ for $i \geq l$ that we have computed. Since the degree of $g_l^{(l)} P_l$ is maximum, we have $\deg(\phi_l) < \deg(g_l^{(l)} P_l)$ if and only if $\deg(S) < \deg(g_l^{(l)} P_l)$ and we can test the latter condition since we have computed these polynomials explicitly. ◀

This algorithm is polynomial in the α_{ij}'s, though ideally we would like it to be polynomial in their bitsize.

▶ **Proposition 18.** *Assume that we have access to an oracle which decides whether*

$$\sum_{i=1}^{k} \prod_{j=1}^{m} a_{ij}^{\alpha_{ij}} = 0. \tag{6}$$

Let $\phi = \sum_{i=1}^{k} \prod_{j=1}^{m} f_j^{\alpha_{ij}}$ as in Theorem 17. Then one can decide deterministically whether ϕ is identically zero in time polynomial in the sparsity of the f_j's and in the bitsize of the a_{ij}'s and α_{ij}'s.

Proof. The only dependency in the α_{ij}'s in the proof of Theorem 17 is the computation of the coefficient of the highest degree monomials of the $g_i^{(l)} P_i$. With the oracle for (6), we skip this step and achieve a polynomial dependency in the bitsize of the α_{ij}'s. ◀

A direct computation of the constant on the left-hand side of (6) is not possible since it involves numbers of exponential bitsize (the exponents α_{ij} are given in binary notation). The test to 0 can be made by computing modulo random primes, but this is ruled out since we want a deterministic algorithm. Note also that this test is a PIT problem for polynomials in $\mathsf{SPS}(k, m, t)$ where the f_j's are constant polynomials. For general arithmetic circuits, it is likewise known that PIT reduces to the case of circuits without any variable occurrence ([4], Proposition 2.2).

The polynomial identity test from Theorem 17 can also be applied to the class of multivariate polynomial families $\mathsf{mSPS}(k, m)$ introduced in the previous section. Indeed, let $P(X_1, \dots, X_n) = \sum_i \prod_j f_j^{\alpha_{ij}}$ belongs to some $\mathsf{mSPS}(k, m)$ family, and suppose we know a bound d on its degree. We turn P into a univariate polynomial Q by the classical substitution (sometimes attributed to Kronecker) $X_i \mapsto X^{(d+1)^i}$. We write $Q(X) = \sum_i \prod_j g_j^{\alpha_{ij}}$, where each univariate polynomial g_j is the image of f_j by the substitution. It is a folklore result that $P \equiv 0$ if and only if $Q \equiv 0$, thus we can apply the PIT algorithm of Theorem 17 on Q.

Let s be the size of the representation of P, meaning that P depends on at most s variables, the f_j's have a constant-free circuit of size at most s and are s-sparse, and the α_{ij} are at most equal to s. (Note that we do not bound their bitsizes but their values as it is needed for our PIT algorithm.) Then the degree of the f_j's is at most 2^s, and $d \leq 2^{\mathsf{poly}(s)}$ where $\mathsf{poly}(s)$ denotes some polynomial function of s. The g_j's therefore have a degree at most $2^{s\mathsf{poly}(s)} \times 2^s = 2^{s\mathsf{poly}(s)+s}$. This proves that Q satisfies the hypothesis of Theorem 17.

5 Conclusion

We have shown that the real τ-conjecture from [14] holds true for a restricted class of polynomials, and from this result we have obtained an identity testing algorithm and a lower bound for the permanent. Other simple cases of the conjecture remain open. In the general case, we can expand a sum of product of sparse polynomials as a sum of at most kt^m monomials. There are therefore at most $2kt^m - 1$ real roots. As pointed out in [14], the

case $k = 2$ is already open: is there a polynomial bound on the number of real roots in this case? Even simpler versions of this question are open. For instance, we can ask whether the number of real roots of an expression of the form $f_1 \cdots f_m + 1$ is polynomial in m and t. A bare bones version of this problem was pointed out by Arkadev Chattopadhyay (personal communication): taking $m = 2$, we can ask what is the maximum number of real roots of an expression of the form $f_1 f_2 + 1$. Expansion as a sum of monomials yields a $O(t^2)$ upper bound, but for all we know the true bound could be $O(t)$.

References

1 M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proceedings of the 25th conference of the Foundations of Software Technology and Theoretical Computer Science*, pages 92–105. Springer, 2005.

2 M. Agrawal and R. Saptharishi. Classifying Polynomials and Identity Testing. In *Current Trends in Science*, pages 149–162. Indian Academy of Sciences, 2009.

3 M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 67–75, 2008.

4 E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. Bro-Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009. Conference version in CCC 2006.

5 M. Beecken, J. Mittmann, and N. Saxena. Algebraic independence and blackbox identity testing. *Proceedings of the 38th International Colloquium on Automata, Languages and Programming*, 2011. Arxiv preprint arXiv:1102.2789.

6 A. Borodin and S. Cook. On the number additions to compute specific polynomials. *SIAM Journal on Computing*, 5(1):146–157, 1976.

7 P. Bürgisser. On defining integers and proving arithmetic circuit lower bounds. *Computational Complexity*, 18(1):81–103, 2009.

8 P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

9 P. Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Algorithms and Computation in Mathematics. Springer, 2000.

10 B. Grenet, P. Koiran, N. Portier, and Y. Strozecki. The Limited Power of Powering: Polynomial Identity Testing and a Depth-four Lower Bound for the Permanent. Manuscript, 2011. `http://arxiv.org/abs/1107.1434`.

11 J. Heintz and C.-P. Schnorr. Testing polynomials which are easy to compute. In *Logic and Algorithmic (an International Symposium held in honour of Ernst Specker)*, pages 237–254. Monographie n^o 30 de L'Enseignement Mathématique, 1982. Preliminary version in *Proc. 12th ACM Symposium on Theory of Computing*, pages 262-272, 1980.

12 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1):1–46, 2004.

13 P. Koiran. Arithmetic circuits: the chasm at depth four gets wider. *Arxiv preprint arXiv:1006.4700*, 2010.

14 P. Koiran. Shallow circuits with high-powered inputs. *Proceedings of the Second Symposium on Innovations in Computer Science*, 2011.

15 T.Y. Li, J.M. Rojas, and X. Wang. Counting real connected components of trinomial curve intersections and m-nomial hypersurfaces. *Discrete and computational geometry*, 30(3):379–414, 2003.

16 N. Saxena. Progress on Polynomial Identity Testing. *Bull. EATCS*, 99:49–79, 2009.

17 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomials identities. *Journal of the ACM*, 27:701–717, 1980.

18 M. Shub and S. Smale. On the intractability of Hilbert's Nullstellensatz and an algebraic version of "P=NP". *Duke Mathematical Journal*, 81(1):47–54, 1995.

19 S. Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20(2):7–15, 1998.

20 L.G. Valiant. Completeness classes in algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 249–261, 1979.

Petri Net Reachability Graphs: Decidability Status of FO Properties

Philippe Darondeau[1], Stéphane Demri[2], Roland Meyer[3], and Christophe Morvan[4]

1 **IRISA/INRIA, Campus de Beaulieu, Rennes, France**
philippe.darondeau@inria.fr
2 **LSV, ENS Cachan, CNRS, INRIA, France** Stephane.Demri@lsv.ens-cachan.fr
3 **University of Kaiserslautern, Germany** meyer@cs.uni-kl.de
4 **Université Paris-Est, Marne-La-Vallée, France**
christophe.morvan@univ-paris-est.fr

Abstract

We investigate the decidability and complexity status of model-checking problems on unlabelled reachability graphs of Petri nets by considering first-order, modal and pattern-based languages without labels on transitions or atomic propositions on markings. We consider several parameters to separate decidable problems from undecidable ones. Not only are we able to provide precise borders and a systematic analysis, but we also demonstrate the robustness of our proof techniques.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, D.2.4 Software/Program Verification

Keywords and phrases Petri nets, First order logic, Reachability graph

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.140

1 Introduction

Decision problems for Petri nets. Much effort has been dedicated to decision problems about Petri nets such as reachability or equivalence, or model checking logical fragments. Reachability is decidable [20] but this is a hard problem. Language equality is, by contrast, undecidable for labelled Petri nets [11, 1]. Many important problems have received decision procedures, e.g., boundedness [16], deadlock-freeness and liveness [10] (by reduction to reachability), semilinearity [12], etc. Hack's thesis [10] provides a comprehensive overview of problems equivalent to reachability. Hack showed that equality of reachability sets of two Petri nets with identical places is undecidable [11]. As our main contribution, we link this result to first-order logic expressing properties of general Petri net reachability graphs.

Our motivations. For Petri nets, model checking CTL formulae with atomic propositions expressing that a place contains at least one token is known to be undecidable [7]. This result carries over to all fragments of CTL containing the modalities EF or AF. Model checking CTL without atomic propositions but with next-time modalities indexed by action labels is undecidable too [7]. In contrast, LTL model-checking over VASS is ExpSpace-complete [9] (atomic propositions are control states). These negative results do not compromise the search for decidable fragments of first-order logic that describe only purely graph-theoretically the reachability graphs. Our intention is to deliberately discard edge labels and atomic propositions on markings. As an example, we consider the structure $(\mathbb{N}^n, \rightarrow)$ derived from a Petri net N with n places such that $M \rightarrow M'$ iff M evolves to M' by firing a transition of N. Since $(\mathbb{N}^n, \rightarrow, =)$ is an automatic structure, its first-order theory is decidable, see e.g. [5].

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 140–151

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, it is unclear what happens if we consider the first-order theory of $\rightarrow$ over the more interesting structure $(\mathrm{Reach}(N), \rightarrow)$. Here, $\mathrm{Reach}(N)$ denotes the set of all markings *reachable* from the initial marking of N. This paper investigates this question and therefore investigates the decidability status of first-order logic with a bit of MSO (via quantification over reachable markings) on $\mathbb{N}^n$, sharing with [21] a common motivation. We study properties of the Petri net reachability graph that are *purely graph-theoretical*; they do not refer to tokens or transition labels and they are mostly *local* in that they can often be expressed in terms of $\rightarrow$ instead of its transitive closure. For instance, this contrasts with logics in [3] that state quantitative properties on markings and transitions, and evaluate formulae on runs.

Our contributions. We investigate the model-checking problem over structures of the form $(\mathrm{Reach}(N), \rightarrow, \xrightarrow{*})$ generated from Petri nets N with first-order languages including predicate symbols for $\rightarrow$ and/or $\xrightarrow{*}$. As it is a classical fragment of first-order logic, we also consider the modal language $\mathrm{ML}(\Box, \Box^{-1})$ with forward and backward modalities. To conclude the study, we consider an alternative framework where the structures are *reachability sets*, subsets of $\mathbb{N}^n$ when the underlying net has n places. For these structures, we study satisfiability of properties defined by *patterns*. Patterns are bounded n-dimensional sets of points that are colored black, white, or grey to mean "reachable", "non-reachable", or "don't care", respectively. Let us mention prominent features of our investigation. (1) Undecidability proofs are obtained by reduction from the equality problem (or the inclusion problem) between reachability sets defined by Petri nets, shown undecidable in [11]. We demonstrate that our proof schema is robust and can be adapted to numerous formalisms specifying local properties as in first-order logic. (2) To determine the cause of undecidability, we investigate logical fragments. At the same time, we strive for maximally expressive decidable fragments. (3) For decidable problems, we assess the computational complexity — either relative to standard complexity classes or by establishing a reduction from the reachability problem for Petri nets. Our main findings are as follows: Model-checking $(\mathrm{Reach}(N), \rightarrow)$ [resp. $(\mathrm{Reach}(N), \xrightarrow{*})$, $(\mathrm{Reach}(N), \xrightarrow{+})$] is undecidable for the appropriate first-order language with one binary predicate symbol. Undecidability is also shown for the positive fragment of $\mathrm{FO}(\rightarrow)$, the forward fragment of $\mathrm{FO}(\rightarrow)$ and $\mathrm{FO}(\rightarrow)$ augmented with $\xrightarrow{*}$ even if the reachability sets are effectively semilinear. We prove that model-checking the existential fragment of $\mathrm{FO}(\rightarrow)$ is decidable, but as hard as the reachability problem for Petri nets. As far as $\mathrm{ML}(\Box, \Box^{-1})$ is concerned, the global model-checking on $(\mathrm{Reach}(N), \rightarrow)$ is undecidable but it becomes decidable when restricted to $\mathrm{ML}(\Box)$ (even if extended with Presburger-definable predicates on markings); the latter problem is also as hard as the reachability problem for Petri nets. The satisfiability of properties defined by bounded patterns is undecidable.

2 Preliminaries

We recall basics on Petri nets and semilinear sets; we introduce Petri net reachability graphs as first-order structures. We define first-order logic and modal logic interpreted on these graphs. Finally, we present decidability results about model-checking problems.

2.1 Petri nets

A *Petri net* is a bi-partite graph $N = (P, T, F, M_0)$, where P and T are *finite* disjoint sets of *places* and *transitions*, and $F : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$. A *marking* of N is a function $M : P \rightarrow \mathbb{N}$. M_0 is the *initial marking* of N. A transition $t \in T$ is *enabled at* a marking

M, written $M[t\rangle$, if $M(p) \geq F(p,t)$ for all places $p \in P$. If t is enabled at M then it can *be fired*. This leads to the marking M' defined by $M'(p) = M(p) + F(t,p) - F(p,t)$ for all $p \in P$, in notation: $M[t\rangle M'$. The definitions are extended to transition sequences $s \in T^*$ in the expected way. A marking M' is *reachable* from a marking M if $M[s\rangle M'$ for some $s \in T^*$. A transition t is *in self-loop* with a place p iff $F(p,t) = F(t,p) > 0$. A transition is *neutral* if it has null effect on all places. The *reachability set* $\mathrm{Reach}(N)$ of N is the set of all markings that are reachable from the initial marking.

▶ **Theorem 2.1.** *(I) [20] Given a Petri net N and two markings M and M', it is decidable whether M' is reachable from M. (II) [11] Given two Petri nets N and N', it is not decidable whether* $\mathrm{Reach}(N) = \mathrm{Reach}(N')$ *[resp.* $\mathrm{Reach}(N) \subseteq \mathrm{Reach}(N')$*].*

A Petri net N induces several standard structures. The *unlabelled reachability graph* of N is the structure $\mathrm{URG}(N) = (D, init, \rightarrow, \xrightarrow{*}, \xrightarrow{+}, =)$ where $D = \mathrm{Reach}(N)$, $init = \{M_0\}$, and $\rightarrow$ is the binary relation on D defined by $M \rightarrow M'$ if $M[t\rangle M'$ for some $t \in T$. The relations $\xrightarrow{*}$ and $\xrightarrow{+}$ are the iterative and strictly iterative closures of $\rightarrow$, respectively. The *reachability graph* $\mathrm{RG}(N)$ of N is $(\mathrm{Reach}(N), \rightarrow)$. The *unlabelled transition graph* of N is the structure $\mathrm{UG}(N) = (D, init, \rightarrow, \xrightarrow{*}, \xrightarrow{+}, =)$ with $D = \mathbb{N}^P$. *Note that reachability of markings is not taken into account in* $\mathrm{UG}(N)$. In the sequel, by default $\mathrm{card}(P) = n$ and we identify $\mathbb{N}^P$ and $\mathbb{N}^n$. We also call 1-*loop* an edge $M \rightarrow M'$ with $M = M'$.
Semilinear subsets of $\mathbb{N}^n$ form an effective Boolean algebra and they coincide with sets definable in Presburger arithmetic (decidable first-order theory of natural numbers with addition). Hence, herein we use equally semilinearity or definability in Presburger arithmetic. Note that in [8], Ginsburg and Spanier gave an effective correspondence between semilinear subsets and subsets of $\mathbb{N}^n$ definable in Presburger arithmetic.We know that given a Petri net N and a semilinear set $E \subseteq \mathbb{N}^{|P|}$ one can decide whether $\mathrm{Reach}(N) \cap E \neq \emptyset$ [11, L. 4.3].

2.2 First-order languages

We introduce a first-order logic FO with atomic predicates $\mathsf{x} \rightarrow \mathsf{y}$, $\mathsf{x} \xrightarrow{*} \mathsf{y}$, $\mathsf{x} \xrightarrow{+} \mathsf{y}$ and $init(\mathsf{x})$. Formulae in FO are defined by $\mathsf{x} \rightarrow \mathsf{y} \mid \mathsf{x} \xrightarrow{*} \mathsf{y} \mid \mathsf{x} \xrightarrow{+} \mathsf{y} \mid init(\mathsf{x}) \mid \mathsf{x} = \mathsf{y} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\, \mathsf{x}\ \varphi \mid \forall\, \mathsf{x}\ \varphi$. Given a set $\mathtt{P}$ of predicate symbols from the above signature, we denote the *restriction of* FO *to the predicates in* $\mathtt{P}$ by $\mathrm{FO}(\mathtt{P})$. Formulae are interpreted either on $\mathrm{URG}(N)$ or on $\mathrm{UG}(N)$. Observe that FO on $\mathrm{UG}(N)$ enables, using *init* and reachability predicates, to relativize formulae to $\mathrm{URG}(N)$. We omit the standard definition of the satisfaction relation $\mathcal{U}, \mathbf{v} \models \varphi$ with $\mathcal{U}$ a structure ($\mathrm{URG}(N)$, $\mathrm{RG}(N)$ or $\mathrm{UG}(N)$) and $\mathbf{v}$ a valuation of the free variables in φ. Typically, $\forall\, \mathsf{x}\ \varphi$ holds true whenever the formula φ holds true for all elements (markings) of the considered structure. *Sentences* are closed formulae, i.e. without free variables. If $\mathcal{U} \models \varphi$ then $\mathcal{U}$ is called a model of φ.

In the sequel, we consider several model-checking problems. The model-checking problem $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO})$ [resp. $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO})$] is stated as follows: given a Petri net N and a sentence $\varphi \in \mathrm{FO}$, does $\mathrm{URG}(N) \models \varphi$ [resp. $\mathrm{UG}(N) \models \varphi$] ? The logics $\mathrm{FO}(\mathtt{P})$ induce restricted model checking problems $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\mathtt{P}))$ and $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(\mathtt{P}))$, respectively. Formulae in FO can express standard structural properties, like deadlock-freeness ($\forall\, \mathsf{x}\ \exists \mathsf{y}\ \mathsf{x} \rightarrow \mathsf{y}$) or cyclicity ($\forall \mathsf{x} \forall \mathsf{y}\ \mathsf{x} \xrightarrow{*} \mathsf{y} \Rightarrow \mathsf{y} \xrightarrow{*} \mathsf{x}$). Semilinear sets and relations are known to be automatic (may be generated by finite synchronous automata [5]). In particular, it means that $(\mathbb{N}^n, \rightarrow, =)$ is automatic. By [5], $(\star)$ $\mathrm{MC}^{\mathcal{S}}(\mathrm{FO})$ is decidable for each automatic structure $\mathcal{S}$. Proposition 2.2 below, consequence of $(\star)$, is our current state of knowledge.

▶ **Proposition 2.2.** (I) $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(\rightarrow,=))$ is decidable. (II) Let $\mathcal{C}$ be a class of Petri nets N for which the restriction on $\mathrm{Reach}(N)$ of the reachability relation $\mathsf{x} \xrightarrow{*} \mathsf{y}$ is effectively semilinear. Then, $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO})$ restricted to $\mathcal{C}$ is decidable. (III) Let $\mathcal{C}$ be a class of Petri nets N for which $\mathrm{Reach}(N)$ is effectively semilinear. Then, $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow,=))$ restricted to $\mathcal{C}$ is decidable.

Here are some classes of Petri nets for which the reachability relation $\xrightarrow{*}$ is effectively semilinear: cyclic Petri nets [2], communication-free Petri nets [6], vector addition systems with states of dimension 2 [18], single-path Petri nets [14], etc.

Note that given φ in $\mathrm{FO}(\rightarrow,=)$, one can effectively build a Presburger formula that characterizes exactly the valuations satisfying φ in $\mathrm{UG}(N)$. However, having $\mathbb{N}^n$ as a domain does not always guarantee decidability, see the undecidability result in [21, Theorem 2] about a structure with domain $\mathbb{N}^n$ but equipped with successor relations for each dimension and with regularity constraints on them.

2.3 Standard first-order fragments: modal languages

By moving along edges, modal languages provide a local view for graph structures. Note the constrast to first-order logic in which one quantifies over any element of the structure. Applications of modal languages include modelling temporal and epistemic reasoning, and they are central for designing logical specification languages. The modal language $\mathrm{ML}(\Box, \Box^{-1})$ (or simply ML) defined below has no propositional variable (like Hennessy-Milner modal logic) and no label on modal operators. This allows us to interpret modal formulae on directed graphs of the form $(\mathrm{Reach}(N), \rightarrow)$. The modal formulae in ML are defined by the grammar $\bot \mid \top \mid \neg\varphi \mid \varphi \wedge \psi \mid \Box\varphi \mid \Box^{-1}\varphi$. We write $\mathrm{ML}(\Box)$ to denote the restriction of ML to $\Box$ and we use the standard abbreviations $\Diamond\varphi \stackrel{\text{def}}{=} \neg\Box\neg\varphi$ and $\Diamond^{-1}\varphi \stackrel{\text{def}}{=} \neg\Box^{-1}\neg\varphi$. We interpret modal formulae on directed graphs $(\mathrm{Reach}(N), \rightarrow)$. We provide the definition of the satisfaction relation $\models$ relatively to an arbitrary directed graph $\mathcal{M} = (W, R)$ and $w \in W$ (clauses for Boolean connectives and logical constants are omitted):

- $\mathcal{M}, w \models \Box\varphi \stackrel{\text{def}}{\Leftrightarrow}$ for every $w' \in W$ such that $(w, w') \in R$, we have $\mathcal{M}, w' \models \varphi$.
- $\mathcal{M}, w \models \Box^{-1}\varphi \stackrel{\text{def}}{\Leftrightarrow}$ for every $w' \in W$ such that $(w', w) \in R$, we have $\mathcal{M}, w' \models \varphi$.

Model-checking problem $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML})$ is the following: given a Petri net N and $\varphi \in \mathrm{ML}$, does $(\mathit{Reach}(N), \rightarrow), M_0 \models \varphi$? Let $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML}(\Box))$ denote $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML})$ restricted to $\mathrm{ML}(\Box)$.

▶ **Proposition 2.3.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML}(\Box))$ is decidable and PSPACE-complete.

Adding $\Box^{-1}$ to $\mathrm{ML}(\Box)$, often does not change the computational complexity of model checking, see e.g. [4]. When it comes to Petri net reachability graphs $\mathrm{RG}(N)$, adding $\Box^{-1}$ preserves decidability but at the cost of performing reachability checks. With a hardness result in Section 3.4, we argue that such checks cannot be avoided.

▶ **Proposition 2.4.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML}(\Box, \Box^{-1}))$ is decidable.

We introduce another decision problem about ML that is related to $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$. The *validity problem* $\mathrm{VAL}^{\mathrm{URG}}(\mathrm{ML})$, is stated as follows: given a Petri net N and $\varphi \in \mathrm{ML}$, does $(\mathrm{Reach}(N), \rightarrow), M \models \varphi$ for every marking $M \in \mathrm{Reach}(N)$? As observed earlier, formulae from $\mathrm{ML}(\Box, \Box^{-1})$ can be viewed as first-order formulae in $\mathrm{FO}(\rightarrow)$. Therefore, using modal languages in specifications is a way to consider fragments of $\mathrm{FO}(\rightarrow)$. Indeed, given φ in $\mathrm{ML}(\Box, \Box^{-1})$, one can compute in linear time a first-order formula φ' with only two individual variables (see e.g. [4]) that satisfies: for every Petri net N we have $\mathrm{RG}(N) \models \varphi'$ iff $\mathrm{RG}(N), M \models \varphi$ for every M in $\mathrm{Reach}(N)$. Hence, the validity problem $\mathrm{VAL}^{\mathrm{URG}}(\mathrm{ML})$ appears as a natural counterpart to $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$.

3 Structural Properties of Unlabelled Net Reachability Graphs

We study the decidability status of model checking unlabelled reachability graphs of Petri nets against the first-order and modal logics defined in the previous section.

3.1 A proof schema for the undecidability of $\mathrm{FO}(\rightarrow)$

To establish undecidability of $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$, we provide a reduction of the equality problem for reachability sets, see Theorem 2.1(II). Given two Petri nets N_1 and N_2 with the same places, we build $\overline{N}$ and φ in $\mathrm{FO}(\rightarrow)$ such that $\mathrm{Reach}(N_1) = \mathrm{Reach}(N_2)$ iff $\mathrm{RG}(\overline{N}) \models \varphi$. Interestingly, φ shall be independent of N_1 and N_2.

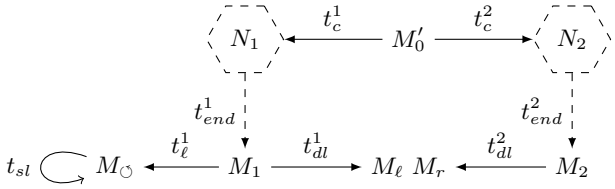


Figure 3.1 Reachability graph of $\overline{N}$

In $\overline{N}$, the nets N_1 and N_2 to be compared for equality of reachability sets share all places except two added control places p_1 and p_2 (set in self-loop with the respective transitions of N_1 and N_2). The Petri net $\overline{N}$ has one more extra place p initially marked. Two concurrent transitions t_c^1 and t_c^2 compete to consume the initial token and mark either p_1 and all places marked in the initial configuration of N_1 or p_2 and all places marked in the initial configuration of N_2 (see Figure 3.1). The first step in the execution of $\overline{N}$ implements an arbitrary choice between simulating N_1 or N_2.

Once the simulation of N_1 or N_2 has started, it may be stopped at any time. This is done by two transitions t_{end}^1 and t_{end}^2 that move the control token from p_1 or p_2 to a new control place p_1' or p_2', thus leading to the marking M_1 or M_2 shown in Figure 3.1. After this, the token count on the places of N_1 and N_2 is not changed any more. Moving the token to p_1' or p_2' switches control to reporting subnets N_1' or N_2' that behave as indicated in Figure 3.1 starting from markings M_1 and M_2.

By just emptying the control place p_1' or p_2', N_1' and N_2' may forget the index 1 or 2 of the net N_1 or N_2 that was simulated and enter a deadlock marking M, that reflects the last marking of N_1 or N_2 in the simulation. For this purpose, $\overline{N}$ is provided with two transitions t_{dl}^1 and t_{dl}^2 (in Figure 3.1, M is denoted M_ℓ and M_r indicating whether it emerged from the simulation of N_1 ($left$) or N_2 ($right$)). $\mathrm{Reach}(N_1) = \mathrm{Reach}(N_2)$ iff every simulation result or deadlock marking M can be obtained from N_1 *and* N_2. But inspecting M in isolation does not reveal whether it stemmed from N_1 or N_2.

Deadlock markings (M) and their immediate predecessor markings (M_1 and/or M_2) are easily characterized by first-order formulae. In order to express in $\mathrm{FO}(\rightarrow)$ that every simulation result M has exactly two direct ancestor markings M_1 and M_2 (such that $M_1[t_{dl}^1\rangle M$ and $M_2[t_{dl}^2\rangle M$), it is necessary that the behaviours of N_1' and N_2' from M_1 or M_2 can be distinguished by $\mathrm{FO}(\rightarrow)$ formulae. For this purpose, one gives to N_1' *but not* to N_2' the possibility to avoid the deadlock state $M_\ell = M$ by firing from M_1 a special transition t_ℓ^1 that leads to a marking ($M_{\circlearrowleft}$) with a 1-loop $t_{\circlearrowleft}$ (no new deadlock is introduced thus). In $\overline{N}$, t_ℓ^1 competes with t_{dl}^1 to move the token from the control place p_1' to another control place

$p_{\circlearrowleft}$, controlling the 1-loop $t_{\circlearrowleft}$. In this way, the formula $\varphi_\ell(\mathsf{x}) \stackrel{\text{def}}{=} \exists\ \mathsf{y}\ (\mathsf{x} \to \mathsf{y} \wedge \mathsf{y} \to \mathsf{y})$ holds in markings M_1 and does not hold in markings M_2.

A formula φ expressing that N_1 and N_2 have equal reachability sets is then: $\forall\ \mathsf{z}\ (\neg\exists \mathsf{z}'\ \mathsf{z} \to \mathsf{z}') \Rightarrow (\exists \mathsf{z}_1\ \mathsf{z}_1 \to \mathsf{z} \wedge \varphi_\ell(\mathsf{z}_1)) \wedge (\exists \mathsf{z}_2\ \mathsf{z}_2 \to \mathsf{z} \wedge \neg\varphi_\ell(\mathsf{z}_2))$. The formula φ requires that for any simulation result M, both logical experiments witnessing for N_1 and N_2 succeed. It is important to observe that the only deadlock markings of $\overline{N}$ are the markings reached by the transitions t^1_{dl} and t^2_{dl}. Lemma 3.1 below, based on this remark, shows that the formula φ expresses in fact the equality of the reachability sets of N_1 and N_2.

The strength of the construction stems from the combination of two ideas. A Petri net can (i) store choices over arbitrary long histories and (ii) reveal this propagated information by finite back and forth experiments determining local structures characterised by first-order formulae. The experiments consist here of one backward transition, reconstructing the initial choice, and some forward transitions checking the presence of a 1-loop.

▶ **Lemma 3.1.** $\mathrm{Reach}(N_1) = \mathrm{Reach}(N_2)$ *if and only if* $\mathrm{RG}(\overline{N}) \models \varphi$.

For the implication from left to right, consider a deadlock marking M. M is only reachable via t^1_{dl} or t^2_{dl}, say $M'_1[t^1_{dl}\rangle M$. Then marking M'_1 satisfies φ_ℓ and stems from a marking $M_1[t^1_{end}\rangle M'_1$ of N_1. The hypothesis on equal reachability sets then yields a marking M_2 of N_2 that leads by transition t^2_{end} to a marking M'_2 satisfying $\neg\varphi_\ell$ as required.

In turn, if φ holds, then we prove two inclusions. To show $\mathrm{Reach}(N_1) \subseteq \mathrm{Reach}(N_2)$, consider marking M_1 reachable via sequence s_1 in N_1. In $\overline{N}$, the marking can be prolonged to a deadlock M with $M'_0[t^1_c\rangle M^1_0[s_1\rangle M_1[t^1_{end}\rangle M'_1[t^1_{dl}\rangle M$. Here, M'_1 satisfies φ_ℓ. But φ yields another predecessor M'_2 of M with $M'_2 \neq M'_1$. To avoid the 1-loop, it has to result from a sequence $M'_0[t^2_c\rangle M^2_0[s_2\rangle M_2[t^2_{end}\rangle M'_2[t^2_{dl}\rangle M$. It is readily checked that M_1 and M_2 coincide up to the token on the control place. This means $M_1 \in \mathrm{Reach}(N_2)$ as required.

By recycling variables in φ above, we get a sharp result that marks the undecidability border of model checking against $\mathrm{FO}(\to)$ by two variables. Model checking $\mathrm{FO}(\to)$ restricted to a one variable is decidable.

▶ **Theorem 3.2.** *There exists a formula* φ *in* $\mathrm{FO}(\to)$ *with two individual variables such that* $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\to))$ *restricted to* φ *is undecidable.*

The above undecidability result can be further sharpened since it is shown in [15] that the undecidability of the equality problem holds already for Petri nets with 5 unbounded places.

3.2 Robustness of the proof schema

Based on the previous proof schema, we present undecidability results for subproblems of $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\to))$. We consider the positive fragment, the forward fragment, the restriction when the direction of edges is omitted, and $\mathrm{ML}(\Box, \Box^{-1})$. Let $\lambda(\mathsf{x}, \mathsf{x}') \stackrel{\text{def}}{=} (\mathsf{x} \to \mathsf{x}') \vee (\mathsf{x}' \to \mathsf{x})$. Expressing properties about $\mathrm{RG}(N)$ in $\mathrm{FO}(\lambda)$ amounts to getting rid of the direction of edges of this graph. Despite this weakening, undecidability is still present. To instantiate the above argumentation, we have to identify deadlock markings and analyse their environment. In $\mathrm{FO}(\lambda)$, we augment markings encountered during the simulation by 3-cycles. Then, the absence of 3-cycles and an environment without such cycles characterises deadlock markings.

▶ **Proposition 3.3.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\lambda))$ is undecidable.

Proposition 3.4 below is proved by adapting the construction depicted in Figure 3.1.

▶ **Proposition 3.4.** $\mathrm{VAL}^{\mathrm{URG}}(\mathrm{ML}(\Box, \Box^{-1}))$ is undecidable.

This undecidability result is tight (see Section 3.3). Translating formulae in $\mathrm{ML}(\Box, \Box^{-1})$ to $\mathrm{FO}(\rightarrow)$ with two individual variables gives another evidence that $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$ with two variables is undecidable. Although $\mathrm{VAL}^{\mathrm{URG}}(\mathrm{ML}(\Box, \Box^{-1}))$ and $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$ are undecidable, we have identified decidable fragments of modal logic in Section 2.3. By analogy, one may expect to find decidable fragments of first-order logic. We prove that this is not the case. We consider here positive $\mathrm{FO}(\rightarrow)$ and forward $\mathrm{FO}(\rightarrow)$. In a *positive formula*, atomic propositions occur only under the scope of an even number of negations. Let $\mathrm{FO}^{+}(\mathtt{P})$ denote the positive fragment of $\mathrm{FO}(\mathtt{P})$. A *forward formula* is a formula in which every occurrence $\mathtt{x} \rightarrow \mathtt{y}$ is in the scope of a quantifier sequence of the form $Q_1\, \mathtt{x} \ldots Q_2\, \mathtt{y}$ where $\mathtt{x}$ is bound before $\mathtt{y}$. Let $\mathrm{FO}_f(\mathtt{P})$ denote the forward fragment of $\mathrm{FO}(\mathtt{P})$.

▶ **Proposition 3.5.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}^{+}(\rightarrow))$ is undecidable.

▶ **Proposition 3.6.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}_f(\rightarrow))$ is undecidable.

While forward formulae can well identify the deadlock markings used in the proof schema, the difficulty is in the description of the local environment witnessing the simulation results.

3.3 Taming undecidability with fragments

In this section, we present the restrictions of $\mathrm{FO}(\rightarrow)$ that we found to have decidable model checking or validity problems. We write $\exists\mathrm{FO}$ for the fragment of FO whose formulae use only existential quantification when written in prenex normal form.

▶ **Proposition 3.7.** $\mathrm{MC}^{\mathrm{URG}}(\exists\mathrm{FO}(\rightarrow, =))$ is decidable.

Decidability of $\mathrm{MC}^{\mathrm{URG}}(\exists\mathrm{FO}(\rightarrow, =))$ is obtained by checking the existence of reachable markings satisfying Presburger constraints. As a corollary, $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow, =))$ restricted to Boolean combinations of existential formulae is decidable, and so is the subgraph isomorphism problem as follows: given a finite directed graph $\mathcal{G}$ and a Petri net N, is there a subgraph of $(\mathrm{Reach}(N), \rightarrow)$ isomorphic to $\mathcal{G}$? Section 3.2 proves that $\mathrm{VAL}^{\mathrm{URG}}(\mathrm{ML}(\Box, \Box^{-1}))$ is undecidable. To our surprise, and in contrast to the negative result on model checking the forward fragment of FO, this undecidability depends on the backward modality, see Proposition 3.8 below (it can be extended to allow labels on edges). We write $\mathrm{PAML}(\Box)$ to denote the extension of $\mathrm{ML}(\Box)$ by allowing as atomic formulae quantifier-free Presburger formulae about the number of tokens in places.

▶ **Proposition 3.8.** The validity problem $\mathrm{VAL}^{\mathrm{URG}}(\mathrm{PAML}(\Box))$ is decidable.

Decidability mainly holds because (non-)satisfaction of formulae in $\mathrm{PAML}(\Box)$ requires the existence of finite tree-like patterns and if the root is in $\mathrm{Reach}(N)$, so are all its nodes (unlike with $\mathrm{ML}(\Box, \Box^{-1})$).

3.4 On the hardness of the decidable problems

Some of our decision procedures call subroutines for solving reachability in Petri nets. As this problem is not known to be primitive recursive, we provide here some complexity-theoretic justification for these costly invocations: we reduce the reachability problem for Petri nets to the decidable problems $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML}(\Box, \Box^{-1}))$ and to $\mathrm{MC}^{\mathrm{URG}}(\exists\mathrm{FO}(\rightarrow))$. Besides reachability, we gave decision procedures that exploit the semilinearity of reachability sets or relations (see e.g. Proposition 2.2), but already for bounded Petri nets, $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$ is of high complexity.

▶ **Proposition 3.9.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$ restricted to bounded Petri nets is decidable but this problem has nonprimitive recursive complexity.

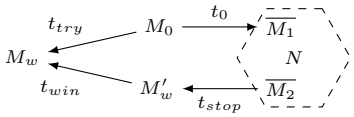


Figure 3.2 Reachability graph in the hardness proof of ML($\Box, \Box^{-1}$)-model checking

▶ **Proposition 3.10.** There is a logarithmic-space reduction from the reachability problem for Petri nets to $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML}(\Box, \Box^{-1}))$.

We reduce reachability of marking M_2 from marking M_1 in a Petri net N to an instance of $\mathrm{MC}^{\mathrm{URG}}(\mathrm{ML}(\Box, \Box^{-1}))$ for a larger net $\overline{N}$. The idea is to introduce a marking M_w (see Figure 3.2) such that the existence of a path to M_w of length greater than 1 witnesses for the existence of some path from M_1 and M_2 in $RG(N)$. To reach M_w by an ML formula, we place it close to the new initial marking. We sketch the argumentation. The inital marking M_0 of $\overline{N}$ contains a single marked place p_i on which compete two transitions t_{try} and t_0. Transition t_{try} moves the unique token from p_i to another place p_w and thus produces the marking M_w *where no other place is marked*. Transition t_0 loads M_1 in the places of N and moves the control token from p_i to another control place p_c set in self-loop with all transitions of N. This starts the simulation of N from M_1. The simulation may be interrupted whenever it reaches a marking of N greater than or equal to M_2. Then, transition t_{stop} consumes M_2 from the places of N and moves the control token from p_c to a place $p_{w'}$. The control token is finally moved from $p_{w'}$ to p_w by firing t_{win}. M_w is reached, after firing $t_{stop}\, t_{win}$, iff $\overline{M_2}$ is reached. Therefore M_2 is reachable from M_1 iff M_w is reachable from $\overline{M_1}$ (its restriction to places of N equals M_1). This is equivalent to stating that M_w has a predecessor different from M_0. The shape of the reachability graph enables to formulate the latter as a local property in $\mathrm{ML}(\Box, \Box^{-1})$: $\varphi := \Diamond(\Box \bot \wedge \Diamond^{-1}\Diamond^{-1}\top)$. Without loss of generality, we can assume that M_1 is no deadlock and $M_2 \neq M_1$. Formula φ requires that M_0 has a deadlock successor and has an incoming path of length two. That the successor is a deadlock means it is not $\overline{M_1}$ but M_w obtained by firing t_{try}. The path from M_0 to M_w is of length one and M_0 has no predecessor. So the path of length two to M_w is not via t_{try} but stems from t_{win}. This means M_w is reachable from $\overline{M_1}$, which means M_2 is reachable from M_1 in N.

The proof of Proposition 3.10 can be adapted to $\exists\mathrm{FO}(\rightarrow)$ for which we also have shown decidability of the model-checking by reduction to the reachability problem for Petri nets.

▶ **Proposition 3.11.** There is a logarithmic-space reduction from the reachability problem for Petri nets to $\mathrm{MC}^{\mathrm{URG}}(\exists\mathrm{FO}(\rightarrow))$.

4 FO with Reachability Predicates

We consider several first-order languages with reachability relations $\xrightarrow{*}$ or $\xrightarrow{+}$, mainly without the one-step relation $\rightarrow$. Undecidability does not follow from Theorem 3.2 since we may exclude $\rightarrow$. Nonetheless we follow the same proof schema. Besides, we distinguish the case when reachability sets are semilinear leading to a surprising undecidability result (Proposition 4.4). Finally, we show that $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(\rightarrow, \xrightarrow{*}))$ is undecidable too.

4.1 FO with reachability relations

The decidability status of $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\xrightarrow{+}))$ is not directly dependent upon the decidability status of $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$. Still we are able to adapt the construction of Section 3.1 but

using now a formula φ in $\mathrm{FO}(\xrightarrow{+})$. The Petri net $\overline{N}$ is the one depicted on Figure 3.1. The formula φ is defined as follows: $\varphi \stackrel{\text{def}}{=} \forall\ \mathsf{z}\ dl(\mathsf{z}) \ \Rightarrow\ (\exists\ \mathsf{z}_1\ (\mathsf{z}_1 \xrightarrow{+} \mathsf{z}) \wedge \varphi_2(\mathsf{z}_1)) \wedge (\exists\ \mathsf{z}_2\ (\mathsf{z}_2 \xrightarrow{+} \mathsf{z}) \wedge \psi_2(\mathsf{z}_2))$ where $dl(\mathsf{z}) \stackrel{\text{def}}{=} \neg\exists \mathsf{z}'\ \mathsf{z} \xrightarrow{+} \mathsf{z}'$, $sl(\mathsf{y}) \stackrel{\text{def}}{=} \mathsf{y} \xrightarrow{+} \mathsf{y}\ \wedge\ \forall \mathsf{w}\ [\mathsf{y} \xrightarrow{+} \mathsf{w} \Rightarrow \mathsf{w} \xrightarrow{+} \mathsf{y}]$, $\varphi_2(\mathsf{z}) \stackrel{\text{def}}{=} [\exists\ \mathsf{y}\ \mathsf{z} \xrightarrow{+} \mathsf{y} \wedge sl(\mathsf{y})] \wedge\ [\forall \mathsf{y}\ \mathsf{z} \xrightarrow{+} \mathsf{y} \Rightarrow (sl(\mathsf{y}) \vee dl(\mathsf{y}))]$, and $\psi_2(\mathsf{z}) \stackrel{\text{def}}{=} [\exists \mathsf{y}\ \mathsf{z} \xrightarrow{+} \mathsf{y} \wedge \forall \mathsf{y}\ \mathsf{z} \xrightarrow{+} \mathsf{y} \Rightarrow dl(\mathsf{y})]$. One can show that $\mathrm{Reach}(N_1) = \mathrm{Reach}(N_2)$ iff $\mathrm{RG}(\overline{N}) \models \varphi$.

▶ **Proposition 4.1.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\xrightarrow{+}))$ is undecidable. Furthermore this results holds for the fixed formula φ defined earlier.

In order to prove undecidability of $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\xrightarrow{*}))$ we have to adapt our usual proof schema, since, in contrast with $\mathrm{FO}(\xrightarrow{+})$, we are no longer able to identify 1-loops.

▶ **Proposition 4.2.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\xrightarrow{*}))$ is undecidable.

Even though $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(\rightarrow, =))$ is decidable (see Proposition 2.2), replacing $\rightarrow$ by $\xrightarrow{*}$ and adding *init* leads to undecidability.

▶ **Corollary 4.3.** $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(init, \xrightarrow{*}))$ *is undecidable.*

Indeed, $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\xrightarrow{*}))$ reduces to $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(init, \xrightarrow{*}))$ by relativization: $\mathrm{URG}(N) \models \varphi$ iff $\mathrm{UG}(N) \models \exists \mathsf{x}_0\ \ init(\mathsf{x}_0) \wedge f(\varphi)$ where φ and $f(\varphi)$ are in $\mathrm{FO}(\xrightarrow{*})$, f is homomorphic for Boolean connectives and $f(\forall\ \mathsf{x}\ \psi) \stackrel{\text{def}}{=} \forall\ \mathsf{x}\ (\mathsf{x}_0 \xrightarrow{*} \mathsf{x}) \Rightarrow f(\psi)$.

4.2 When semilinearity enters into the play

We saw that $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow, =))$ restricted to Petri nets with effectively semilinear reachability sets is decidable (see Proposition 2.2), but it is unclear what happens if the relation $\xrightarrow{*}$ is added. We establish that $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow, \xrightarrow{*}))$ restricted to Petri nets with semilinear reachability sets is undecidable, by a reduction from $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow))$. Given a Petri net N and a sentence $\varphi \in \mathrm{FO}(\rightarrow)$, we reduce the truth of φ in $\mathrm{RG}(N)$ to the truth of a formula $\overline{\varphi}$ in $\mathrm{RG}(\overline{N})$ with a semilinear reachability set. The Petri net $\overline{N}$ is defined from N by adding the new places p_0, p_1 and p_2; each transition from N is in self-loop with p_1. Moreover, we add a new set of transitions that are in self-loop with p_2 and that consist in adding or removing tokens from the original places of N (thus modifying its content arbitrarily). These transitions form a subnet denoted by Br. Three other transitions are added; see Figure 4.1 for a schematic representation of $\overline{N}$ (initial marking M_0' of $\overline{N}$ restricted to places in N is M_0 with $M_0'(p_0) = M_0'(p_1) = 1$ and $M_0'(p_2) = 0$). Our intention is to enforce $\mathrm{Reach}(\overline{N})$ to be semilinear while being able to identify a subset from $\mathrm{Reach}(\overline{N})$ that is in bijection with $\mathrm{Reach}(N)$; this is a way to drown $\mathrm{Reach}(N)$ into $\mathrm{Reach}(\overline{N})$. Indeed, $\mathrm{Reach}(\overline{N})$ contains all the markings such that the sum of p_1 and p_2 is 1 and p_0 is at most 1. Moreover, if the transition t is fired first, then the subsequently reachable markings are precisely those of N; $\mathrm{RG}(N)$ embeds isomorphically into $\mathrm{RG}(\overline{N})$. Until t is fired, one may always come back to M_0', using the brownian subnet Br, but this is impossible afterwards.

▶ **Proposition 4.4.** $\mathrm{MC}^{\mathrm{URG}}(\mathrm{FO}(\rightarrow, \xrightarrow{*}))$ restricted to Petri nets with semilinear reachability sets is undecidable.

Proof. In a first stage, we use *init* although this predicate cannot be expressed in $\mathrm{FO}(\rightarrow, \xrightarrow{*})$. Let $\overline{\varphi}$ be the formula $\exists\ \mathsf{x}_0\ \mathsf{x}_1\ init(\mathsf{x}_0) \wedge \mathsf{x}_0 \rightarrow \mathsf{x}_1 \wedge \neg(\mathsf{x}_1 \xrightarrow{*} \mathsf{x}_0) \wedge f(\varphi)$ where $f(\cdot)$ is homomorphic for Boolean connectives and $f(\forall\ \mathsf{x}\ \psi) \stackrel{\text{def}}{=} \forall\ \mathsf{x}\ (\mathsf{x}_1 \xrightarrow{*} \mathsf{x}) \Rightarrow f(\psi)$ (relativization). In $\overline{\varphi}$, x_0 is interpreted as the initial marking, and x_1 is interpreted as a successor of x_0 from which x_0 cannot be reached again. This may only happen by firing t from M_0'. Now the relativization of every other variable to x_1 in $\overline{\varphi}$ ensures that $\mathrm{RG}(N) \models \varphi$ iff $\mathrm{RG}(\overline{N}) \models \overline{\varphi}$. To remove

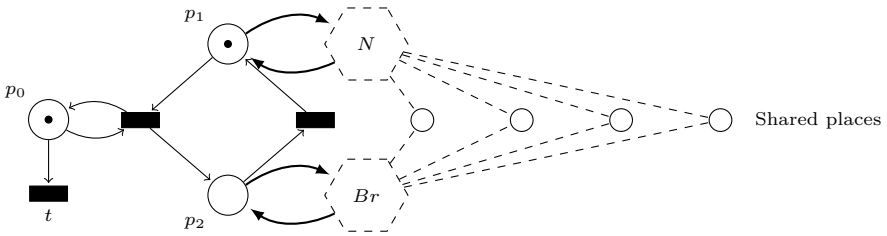


Figure 4.1 Petri net $\overline{N}$

init, we construct a Petri net $\overline{N}'$ similar to $\overline{N}$. $\overline{N}'$ has an extra place p'_0, initially marked with one token, and a new transition that consumes this token and produces two tokens in p_0 and p_1, which were initially empty. By construction, the initial marking of $\overline{N}'$ is the sole marking in $\mathrm{RG}(\overline{N}')$ with no incoming edge and one outgoing edge. We use the formula $\overline{\varphi}' = \exists\, \mathsf{x}'_0\ \mathsf{x}_0\ \mathsf{x}_1\ (\neg\exists\, \mathsf{y}\ \mathsf{y} \to \mathsf{x}'_0)\ \wedge\ \mathsf{x}'_0 \to \mathsf{x}_0\ \wedge\ \mathsf{x}_0 \to \mathsf{x}_1 \wedge (\neg \mathsf{x}_1 \xrightarrow{*} \mathsf{x}_0) \wedge f(\varphi)$. For the same reasons as above, $\mathrm{RG}(N) \models \varphi$ iff $\mathrm{RG}(\overline{N}') \models \overline{\varphi}'$. ◀

4.3 The reachability relation and the structure UG

Corollary 4.3 states a first undecidable result for UG. In this section we examine two other cases where the model checking of formulas in $\mathrm{FO}(\to, \xrightarrow{*})$ are undecidable for this structure.

▶ **Proposition 4.5.** $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(\to, \xrightarrow{*}))$ is undecidable.

Proposition 4.5 holds even when the reachable set of the net is effectively semilinear.

▶ **Proposition 4.6.** $\mathrm{MC}^{\mathrm{UG}}(\mathrm{FO}(\to, \xrightarrow{*}))$ is undecidable for classes of Petri nets having an effective semilinear reachability set.

In this section we have examined several first-order sublanguages involving the reachability predicate. We obtained undecidability results, even when the reachable markings form a semilinear set, and even when $\mathrm{UG}(N)$ is considered instead of $\mathrm{URG}(N)$

5 Pattern Matching Problem

In this section, we do not consider the reachability graphs of Petri nets but their reachability sets ($\mathrm{Reach}(N)$), plain subsets of $\mathbb{N}^n$ where n is the number of places of the net. In [17] the author characterizes such sets as *almost-semilinear* sets, a global property. On the opposite, we focus here on the shape of local neighborhoods by determining the existence of markings in $\mathbb{N}^n$ whose surrounding satisfies a specific pattern of reachable and non-reachable positions.

Using such patterns, one may check for instance whether there exist two reachable markings that differ only on a fixed place and by exactly one token.

A *pattern* $\mathcal{P}$ is defined as a map $[0, N_1] \times \cdots \times [0, N_n] \to \{\{\circ\}, \{\bullet\}, \{\circ, \bullet\}\}$ (values 'unreachable', 'reachable', 'dontcare'). A *constrained* position for $\mathcal{P}$ is an element of $[0, N_1] \times \cdots \times [0, N_n]$ with $\mathcal{P}$-image different from $\{\circ, \bullet\}$. Observe that patterns have the full dimension of the state space of the net. Each Petri net N with n (ordered) places induces a map $f_N : \mathbb{N}^n \to \{\{\circ\}, \{\bullet\}\}$ such that $f_N(M) = \{\bullet\}$ iff $M \in \mathrm{Reach}(N)$. Given a Petri net N, a pattern $\mathcal{P}$ is *matched* by the net N at a point $\vec{v} \in \mathbb{N}^n$ if, for all $\vec{a} \in [0, N_1] \times \cdots \times [0, N_n]$, $f_N(\vec{v} + \vec{a}) \subseteq \mathcal{P}(\vec{a})$. A pattern $\mathcal{P}$ is *matched* by a Petri net N if it is matched by N at some point $\vec{v} \in \mathbb{N}^n$ (that may not be a reachable marking). The *Pattern Matching Problem* (PMP) is defined as follows: given a Petri net N and a pattern $\mathcal{P}$, is $\mathcal{P}$ matched by N?

Table 1 Summary

Problem	$\sharp$	Arbitrary	Effectively semilinear Reach(N)
MC$^\sharp$(FO($\rightarrow$))	URG UG	**UNDEC** (Theo. 3.2) DEC	**DEC** DEC
MC$^\sharp$(FO($\xrightarrow{+}$))	URG	**UNDEC** (Prop. 4.1)	open
MC$^\sharp$(FO($\xrightarrow{*}$))	URG	**UNDEC** (Prop. 4.2)	open
MC$^\sharp$(FO($\rightarrow, \xrightarrow{*}$))	URG UG	UNDEC **UNDEC** (Prop. 4.5)	**UNDEC** (Prop. 4.4) **UNDEC** (Prop. 4.6)
MC$^\sharp$(FO$^+$($\rightarrow$))	URG	**UNDEC** (Prop. 3.5)	DEC
MC$^\sharp$(FO$_f$($\rightarrow$))	URG	**UNDEC** (Prop. 3.6)	DEC
MC$^\sharp$($\exists$FO($\rightarrow, =$))	URG	**DEC** (Prop. 3.7)	DEC
MC$^\sharp$(FO($\rightarrow, =$))	UG	**DEC** (Prop. 2.2)	DEC
MC$^\sharp$(ML($\Box$))	URG	**PSpace-complete**	PSPACE-complete
MC$^\sharp$(ML($\Box, \Box^{-1}$))	URG	**DEC** (Prop. 2.4)	DEC
VAL$^\sharp$(ML($\Box, \Box^{-1}$))	URG	**UNDEC** (Prop. 3.4)	DEC
VAL$^\sharp$(PAML($\Box$))	URG	**DEC** (Prop. 3.8)	DEC
PMP	$\mathbb{N}^n$	**UNDEC** (Proposition 5.1)	**DEC** (Proposition 5.1)

▶ **Proposition 5.1.** (1) Let $\mathcal{C}$ be a class of Petri nets with effectively semilinear reachability sets. Then, PMP restricted to Petri nets in $\mathcal{C}$ is decidable. (2) PMP restricted to patterns with at most two constrained positions is undecidable.

Proposition 5.1(1) follows from the semilinearity of the set of marking satisfying patterns. To prove (2) we embed the reachable sets of two nets into two hyperplanes. Then these sets do not match iff there are two markings one reachable, the other not which may be encoded into a pattern. We use, here, a pattern with 2 adjacent, reachable and non-reachable, positions. It seems uneasy to prove this result using patterns having a single kind of constraints.

6 Concluding Remarks

We investigated mainly the model-checking problem over unlabelled reachability graphs of Petri nets with FO($\rightarrow$). The robustness of our main undecidability proof has been tested against standard fragments of FO($\rightarrow$), modal fragments, patterns and against the additional assumption that reachability sets are effectively semilinear. Table 1 provides a summary of the main results (observe that whenever the reachability relation is effectively semilinear, each problem is decidable). Results in bold are proved in the paper, whereas unbold ones are their consequences. Despite the quantity of results, a few rules of thumb can be synthesized: (1) undecidability of MC(FO($\rightarrow$)) is robust for several fragments of FO($\rightarrow$); (2) decidability results with simple restrictions such as considering bounded Petri nets or $\exists$FO($\rightarrow$) lead to computationally difficult problems (see Section 3.4); (3) the above points are still relevant for modal languages and patterns. Let us conclude by mentioning possible continuations of this work. Firstly, our taxonomy of results is partially incomplete.

New directions can also be followed. First, one could check geometrical properties of the reachability set Reach(N), e.g., the existence of an homogeneous ball around some reachable marking. Second, one could ask decidability questions about infinite unfoldings of nets in place of net reachability graphs. Such unfoldings may be shaped as trees if they may be local event structures [13]. With tree-unfoldings, labelling arcs (or nodes) is required if one wants to be able to express non-trivial properties, but then markings can be encoded to trees in which each arc represents one token being removed from a place identifies by the label of the arc. With event structure unfoldings, labelling an event e by a (sufficiently large)

number k may always be simulated by adding k events triggered by e and in direct conflict with one another. In both cases, for obtaining decidable fragments of FO, one must avoid introducing any relation that would allow comparing for isomorphism two subtrees of two substructures triggered by two different events (like t^1_{dl} and t^2_{dl} in Fig. 3.1). The situation is different with *regular* trace event structures, although the substructure triggered by an event is characterized here by the label of this event. The decidability of FO over regular trace event structures has indeed been shown in [19]. However, *regular* trace event structures model safe Petri nets, whereas the model checking questions studied in this paper bear upon general and thus unbounded Petri nets.

Acknowledgments: We would like to thank the anonymous referees for helpful remarks and suggestions.

References

1 T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *TCS*, 3:85–104, 1976.

2 T. Araki and T. Kasami. Decidability problems on the strong connectivity of Petri net reachability sets. *TCS*, 4:99–119, 1977.

3 M. F. Atig and P. Habermehl. On Yen's path logic for Petri nets. *International Journal of Foundations of Computer Science*, 22(4):783–799, 2011.

4 P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. CUP, 2001.

5 A. Blumensath and E. Grädel. Automatic structures. In *LICS'00*, pages 51–62, 2000.

6 J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(13):13–26, 1997.

7 J. Esparza. Decidability and complexity of Petri net problems — an introduction. In *Advances in Petri Nets 1998*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998.

8 S. Ginsburg and E. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

9 P. Habermehl. On the complexity of the linear-time mu-calculus for Petri nets. In *ICATPN'97*, volume 1248 of *LNCS*, pages 102–116. Springer, 1997.

10 M. Hack. *Decidability Questions for Petri nets*. PhD thesis, MIT, 1975.

11 M. Hack. The equality problem for vector addition systems is undecidable. *TCS*, 2:77–96, 1976.

12 D. Hauschildt. Semilinearity of the reachability set is decidable for Petri nets. Technical Report FBI-HH-B-146/90, University of Hamburg, 1990.

13 P. W. Hoogers, H. C. M. Kleijn, and P. S. Thiagarajan. An event structure semantics for general Petri nets. *TCS*, 153:129–170, 1993.

14 R. Howell, P. Jančar, and L. Rosier. Completeness results for single-path Petri nets. *I & C*, 106(2):253–265, 1993.

15 P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *TCS*, 148:281–301, 1995.

16 R. M. Karp and R. E. Miller. Parallel program schemata. *JCSS*, 3:147–195, 1969.

17 J. Leroux. Vector Addition System Reachability Problem (A Short Self-Contained Proof). In *POPL'11*, pages 307–316, 2011.

18 J. Leroux and G. Sutre. On Flatness for 2-Dimensional Vector Addition Systems with States. In *CONCUR'04*, volume 3170 of *LNCS*, pages 402–416. Springer, 2004.

19 P. Madhusudan. Model-checking trace event structures. In *LICS'03*, pages 371–380, 2003.

20 E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13(3):441–460, 1984.

21 S. Schulz. First-order logic with reachability predicates on infinite systems. In *FST&TCS'10*, pages 493–504. LIPICS, 2010.

Approximating Petri Net Reachability Along Context-free Traces

Mohamed Faouzi Atig[1] and Pierre Ganty[2]

1 **Uppsala University, Sweden**
mohamed_faouzi.atig@it.uu.se
2 **IMDEA Software Institute, Spain**
pierre.ganty@imdea.org

Abstract

We investigate the problem asking whether the intersection of a context-free language (CFL) and a Petri net language (PNL) (with reachability as acceptance condition) is empty. Our contribution to solve this long-standing problem which relates, for instance, to the reachability analysis of recursive programs over unbounded data domain, is to identify a class of CFLs called the finite-index CFLs for which the problem is decidable. The k-index approximation of a CFL can be obtained by discarding all the words that cannot be derived within a budget k on the number of occurrences of non-terminals. A finite-index CFL is thus a CFL which coincides with its k-index approximation for some k. We decide whether the intersection of a finite-index CFL and a PNL is empty by reducing it to the reachability problem of Petri nets with *weak* inhibitor arcs, a class of systems with infinitely many states for which reachability is known to be decidable. Conversely, we show that the reachability problem for a Petri net with weak inhibitor arcs reduces to the emptiness problem of a finite-index CFL intersected with a PNL.

1998 ACM Subject Classification D.2.4 Software Engineering /Program Verification

Keywords and phrases Petri nets, Context-free Grammars, Reachability Problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.152

1 Introduction

Automated verification of infinite-state systems, for instance programs with (recursive) procedures and integer variables, is an important and a highly challenging problem. Pushdown automata (or equivalently context-free grammars) have been proposed as an adequate formalism to model procedural programs. However pushdown automata require finiteness of the data domain which is typically obtained by abstracting the program's data, for instance, using the predicate abstraction techniques [3, 9]. In many cases, reasoning over finite abstract domains leads to too coarse an analysis and is therefore not precise. To palliate this problem, it is natural to model a procedural program with integer variables as a pushdown automaton manipulating counters. In general, pushdown automata with counters are Turing powerful which implies that basic decision problems are undecidable (this is true even for the case finite-state automata with counters).

Therefore one has to look for restrictions on the model which retain sufficient expressiveness while allowing basic properties like reachability to be algorithmically verified. One such restriction is to forbid the test of a counter and a constant for equality. In fact, forbidding test for equality implies the decidability of the reachability problem for the case of finite-state automata with counters (i.e. Petri nets [13, 15]).

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 152–163

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The verification problem for pushdown automata with (restricted) counters boils down to check whether a context-free language (CFL) and a Petri net language (PNL) (with reachability as acceptance condition) are disjoint or not. We denote this last problem $\mathsf{PNL} \cap \mathsf{CFL} \stackrel{?}{=} \emptyset$.

The decidability of $\mathsf{PNL} \cap \mathsf{CFL} \stackrel{?}{=} \emptyset$ is open and lies at the very edge of our comprehension of infinite-state systems. We see two breakthroughs contributing to this question. First, determining the emptiness of a PNL was known to be decidable as early as the eighties. Then, in 2008, Reinhardt [15] lifted this result to an extension of PN with inhibitor arcs (that allow to test if a counter equals 0) which must satisfy some additional topological conditions. By imposing a topology on the tests for zero, Reinhardt prevents his model to acquire Turing powerful capabilities. We call his model PNW and their languages PNWL.

Our contribution to the decidability of $\mathsf{PNL} \cap \mathsf{CFL} \stackrel{?}{=} \emptyset$ comes under the form of a partial answer which is better understood in terms of underapproximation. In fact, given a PNL L_1 and a language L of a context-free grammar, we under-approximate L by a subset L' which is obtained by discarding from L all the words that cannot be derived within a given budget $k \in \mathbb{N}$ on the number of non-terminal symbols. (In fact, the subset L' contains any word of L that can be generated by a derivation of the context-free grammar that contains at most k non-terminal symbols at each derivation step.) We show how to compute L' by annotating the variables of the context-free grammar for L with an allowance. What is particularly appealing is that the coverage of L increases with the allowance. Approximations induced by allowances are non-trivial: every regular or linear language is captured exactly with an allowance of 1, L' coincides with L when the allowance is unbounded, and under commutativity of concatenation L' coincides with L for some allowance $k \in \mathbb{N}$.

We call finite-index CFL, or fiCFL for short, a context-free language where each of its words can be derived within a given budget. In this paper, we prove the decidability of $\mathsf{PNL} \cap \mathsf{fiCFL} \stackrel{?}{=} \emptyset$ by reducing it to the emptiness problem of PNWL. We also prove the converse reduction; showing those two problems are equivalent. Hence, we offer a whole new perspective on the emptiness problem for PNWL and $\mathsf{PNL} \cap \mathsf{CFL}$.

To conclude the introduction let us mention the recent result of [2] which builds on [13] to give an alternative proof of Reinhardt's result (PNW reachability is decidable) for the particular case where one counter only can be tested for zero.

2 Preliminaries

2.1 Context-Free Languages

An *alphabet* Σ is a finite non-empty set of *symbols*. A *word* w over an alphabet Σ is a finite sequence of symbols of Σ where the empty sequence is denoted ε. We write Σ^* for the set of words over Σ. Let $L \subseteq \Sigma^*$, L defines a *language*.

A *context-free grammar* (CFG) G is a tuple $(\mathcal{X}, \Sigma, \mathcal{P})$ where $\mathcal{X}$ is a finite non-empty set of *variables* (*non-terminal letters*), Σ is an alphabet of *terminal letters*, and $\mathcal{P} \subseteq \big(\mathcal{X} \times (\mathcal{X}^2 \cup \Sigma \cup \{\epsilon\})\big)$ is a finite set of *productions* (the production (X, w) may also be denoted by $X \to w$). For every production $p = (X, w) \in \mathcal{P}$, we use $head(p)$ to denote the variable X. Observe that the form of the productions is restricted, but it has been shown in [12] that every CFG can be transformed, in polynomial time, into an equivalent grammar of this form.

Given two strings $u, v \in (\Sigma \cup \mathcal{X})^*$ we define the relation $u \Rightarrow v$, if there exists a production $(X, w) \in \mathcal{P}$ and some words $y, z \in (\Sigma \cup \mathcal{X})^*$ such that $u = yXz$ and $v = ywz$. We use $\Rightarrow^*$ for the reflexive transitive closure of $\Rightarrow$. Given $X \in \mathcal{X}$, we define the language $L_G(X)$, or

simply $L(X)$ when G is clear form the context, as $\{w \in \Sigma^* \mid X \Rightarrow^* w\}$. A language L is *context-free* (CFL) if there exists a CFG $G = (\mathcal{X}, \Sigma, \mathcal{P})$ and $A \in \mathcal{X}$ such that $L = L_G(A)$.

2.2 Finite-index Approximation of Context-Free Languages

Let $k \in \mathbb{N}$, $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a CFG and $A \in \mathcal{X}$. A derivation from A given by $A = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n$ is *k-index bounded* if for every $i \in \{0, \ldots, n\}$ at most k symbols of α_i are variables. We denote by $L^{(k)}(A)$ the subset of $L(A)$ such that for every $w \in L^{(k)}(A)$ there exists a k-index bounded derivation $A \Rightarrow^* w$. We call $L^{(k)}(A)$ the *k-index approximation* of $L(A)$ or more generically we say that $L^{(k)}(A)$ is a *finite-index approximation* of $L(A)$.[1]

Let us now give some known properties of finite-index approximations. Clearly $\lim_{k\to\infty} L^{(k)}(A) = L(A)$. Moreover, let L be a regular or linear language[2], then there exists a CFG G', and a variable A' of G' such that $L(A') = L = L^{(1)}(A')$. Also Luker showed in [14] that if $L(A) \subseteq L(w_1^* \cdots w_n^*)$ for some $w_i \in \Sigma^*$, then $L^{(k)}(A) = L(A)$ for some $k \in \mathbb{N}$. More recently, [6, 8] showed some form of completeness for finite-index approximation when commutativity of concatenation is assumed. It shows that there exists a $k \in \mathbb{N}$ such that $L(A) \subseteq \Pi(L^{(k)}(A))$ where $\Pi(L)$ denotes the language obtained by permuting symbols of w for every $w \in L$. As an incompleteness result, Salomaa showed in [16] that for the Dyck language $L_{D_1^*}$ over 1-pair of parentheses there is no CFG G', variable A' of G' and $k \in \mathbb{N}$ such that $L^{(k)}(A') = L_{D_1^*}$.

Inspired by [5, 7, 6] let us define the CFG $G^{[k]}$ which annotates the variables of $\mathcal{X}$ with a positive integer. With this annotation we can capture precisely finite-index approximations of $L(A)$ as given in Lem. 2.

▶ **Definition 1.** Let $G^{[k]} = (\mathcal{X}^{[k]}, \Sigma, \mathcal{P}^{[k]})$ be the context-free grammar defined as follows: $\mathcal{X}^{[k]} = \left\{X^{[i]} \mid 0 \leq i \leq k \wedge X \in \mathcal{X}\right\}$, and $\mathcal{P}^{[k]}$ is the smallest set such that:

- For every $X \to Y\,Z \in \mathcal{P}$, $\mathcal{P}^{[k]}$ has the productions $X^{[i]} \to Y^{[i-1]}Z^{[i]}$ and $X^{[i]} \to Y^{[i]}Z^{[i-1]}$ for every $i \in \{1, \ldots, k\}$.
- For every $X \to \sigma \in \mathcal{P}$ with $\sigma \in \Sigma \cup \{\epsilon\}$, $X^{[i]} \to \sigma \in \mathcal{P}^{[k]}$ for all $i \in \{0, \ldots, k\}$.

What follows is a consequence of several results from different papers by Esparza *et al.* Because of space constraints the proof is given in [1].

▶ **Lemma 2.** *Let $X \in \mathcal{X}$. We have $L(X^{[k]}) = L^{(k+1)}(X)$.*

2.3 Petri nets with Inhibitor Arcs

Let Σ be a finite non-empty set, a *multiset* (or a marking) $\mathbf{m} \colon \Sigma \to \mathbb{N}$ over Σ maps each symbol of Σ to a natural number. Let $\mathbb{M}[\Sigma]$ be the set of all multisets over Σ.

Sometimes, we use $\mathbf{m} = [\![q_1, q_1, q_3]\!]$ to denote the multiset $\mathbf{m} \in \mathbb{M}[\{q_1, q_2, q_3, q_4\}]$ such that $\mathbf{m}(q_1) = 2$, $\mathbf{m}(q_2) = \mathbf{m}(q_4) = 0$, and $\mathbf{m}(q_3) = 1$. The empty multiset is denoted $\varnothing$.

Given $\mathbf{m}, \mathbf{m}' \in \mathbb{M}[\Sigma]$ we define $\mathbf{m} \oplus \mathbf{m}' \in \mathbb{M}[\Sigma]$ to be the multiset such that $\forall a \in \Sigma \colon (\mathbf{m} \oplus \mathbf{m}')(a) = \mathbf{m}(a) + \mathbf{m}'(a)$, we also define the natural partial order $\preceq$ on $\mathbb{M}[\Sigma]$ as follows: $\mathbf{m} \preceq \mathbf{m}'$ iff there exists $\mathbf{m}^{\Delta} \in \mathbb{M}[\Sigma]$ such that $\mathbf{m} \oplus \mathbf{m}^{\Delta} = \mathbf{m}'$. We also define $\mathbf{m} \ominus \mathbf{m}' \in \mathbb{M}[\Sigma]$ as the multiset such that $(\mathbf{m} \ominus \mathbf{m}') \oplus \mathbf{m}' = \mathbf{m}$ provided $\mathbf{m}' \preceq \mathbf{m}$.

A *Petri net* with inhibitor arcs (PNI for short) $N = (S, T, F = \langle Z, I, O\rangle, \mathbf{m}_\iota)$ consists of a finite non-empty set S of *places*, a finite set T of *transitions* disjoint from S, a tuple

[1] Finite-index approximations were first studied in the 60's.
[2] See [11] for definitions.

$F = \langle Z, I, O\rangle$ of functions $Z\colon T \to 2^S$, $I\colon T \to \mathbb{M}[S]$ and $O\colon T \to \mathbb{M}[S]$, and an *initial marking* $\mathbf{m}_\iota \in \mathbb{M}[S]$. A marking $\mathbf{m}$ ($\in \mathbb{M}[S]$) of N assigns to each place $p \in S$ $\mathbf{m}(p)$ *tokens*.

A transition $t \in T$ is *enabled at* $\mathbf{m}$, written $\mathbf{m}\,[t\rangle$, if $I(t) \preceq \mathbf{m}$ and $\mathbf{m}(p) = 0$ for all $p \in Z(t)$. A transition t that is enabled at $\mathbf{m}$ can be *fired*, yielding a marking $\mathbf{m}'$ such that $\mathbf{m}' = (\mathbf{m} \ominus I(t)) \oplus O(t)$. We write this fact as follows: $\mathbf{m}\,[t\rangle\,\mathbf{m}'$. We extend enabledness and firing inductively to finite sequences of transitions as follows. Let $w \in T^*$. If $w = \varepsilon$ we define $\mathbf{m}\,[w\rangle\,\mathbf{m}'$ iff $\mathbf{m}' = \mathbf{m}$; else if $w = u \cdot v$ we have $\mathbf{m}\,[w\rangle\,\mathbf{m}'$ iff $\exists \mathbf{m}_1 \colon \mathbf{m}\,[u\rangle\,\mathbf{m}_1 \wedge \mathbf{m}_1\,[v\rangle\,\mathbf{m}'$.

A marking $\mathbf{m} \in \mathbb{M}[S]$ is *reachable from* $\mathbf{m}_0$ if and only if there exists $w \in T^*$ such that $\mathbf{m}_0\,[w\rangle\,\mathbf{m}$. Given a language $L \subseteq T^*$ over the transitions of N, the *set of reachable markings from* $\mathbf{m}_0$ *along* L, written $[\mathbf{m}_0\rangle^L$, is defined by $\{\mathbf{m} \mid \exists w \in L \colon \mathbf{m}_0\,[w\rangle\,\mathbf{m}\}$. Incidentally, if L is unspecified then it is assumed to be T^* and we simply write $[\mathbf{m}_0\rangle$ for the set of markings reachable from $\mathbf{m}_0$. To avoid ambiguities, we sometimes explicit the PNI, e.g. $\mathbf{m}_1 \in [\mathbf{m}_0\rangle^L_N$.

A Petri net with *weak* inhibitor arcs (PNW for short) is a PNI $N = (S, T, F = \langle Z, I, O\rangle, \mathbf{m}_\iota)$ such that there is an index function $f\colon S \to \mathbb{N}$ with the property:

$$\forall p, p' \in S \colon f(p) \leq f(p') \to (\forall t \in T \colon p' \in Z(t) \to p \in Z(t)) \ . \tag{1}$$

A Petri net (PN for short) can be seen as a subclass of Petri nets with *weak* inhibitor arcs where $Z(t) = \emptyset$ for all transitions $t \in T$. In this case, we shorten F as the pair $\langle I, O\rangle$.

The *reachability* problem for a PNI $N = (S, T, F = \langle Z, I, O\rangle, \mathbf{m}_\iota)$ is the problem of deciding, for a given marking $\mathbf{m}$, whether $\mathbf{m} \in [\mathbf{m}_\iota\rangle$ holds. It is well known that reachability for Petri nets with inhibitor arcs is undecidable [10]. However, the following holds:

▶ **Theorem 3.** *[15] The reachability problem for* PNW *is decidable.*

2.4 The reachability problem for Petri nets along finite-index CFL

Let us formally define the problem we are interested in. Given: (1) a Petri net $N = (S, T, F, \mathbf{m}_\iota)$ where $T \neq \emptyset$; (2) a CFG $G = (\mathcal{X}, T, \mathcal{P})$ and $A \in \mathcal{X}$; (3) a marking $\mathbf{m}_f \in \mathbb{M}[S]$; and (4) a value $k \in \mathbb{N}$.

$$\text{Does } \mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L^{(k)}(A)} \text{ hold ?}$$

In what follows, we prove the interreducibility of the reachability problem for PN along finite-index CFL and the reachability problem for PNW.

3 From PN reachability along fiCFL to PNW reachability

In this section, we show that the reachability problem for Petri nets along finite-index CFL is decidable. To this aim, let us fix an instance of the problem: a Petri net $N = (S, T, F, \mathbf{m}_\iota)$ where $T \neq \emptyset$, a CFG $G = (\mathcal{X}, T, \mathcal{P})$, $\mathbf{m}_f \in \mathbb{M}[S]$, and a natural number $k \in \mathbb{N}$. Moreover, let $G^{[k]} = (\mathcal{X}^{[k]}, T, \mathcal{P}^{[k]})$ be the CFG given by def. 1.

Lemma 2 shows that $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L^{(k+1)}(A)}$ if and only if $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L(A^{[k]})}$. Then, our decision procedure, which determines if $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L(A^{[k]})}$, proceeds by reduction to the reachability problem for PNW and is divided in two steps. First, we reduce the question $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L(A^{[k]})}$ to the existence of a successful execution in the program of Alg. 1 which, in turn, is reduced to a reachability problem for PNW. Let us describe Alg. 1.

Part 1. Alg. 1 gives the procedure *traverse* in which $\mathbf{M}_\mathsf{i}$ and $\mathbf{M}_\mathsf{f}$ are global arrays of markings with index ranging from 0 to k (i.e., for every $j \in \{0,\dots,k\}$, $\mathbf{M}_\mathsf{i}[j], \mathbf{M}_\mathsf{f}[j] \in \mathbb{M}[S]$). We say that a call $traverse(X^{[\ell]})$ *successfully returns* if there exists an execution which eventually reaches line 22 (i.e., no assert fails) and the postcondition $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for every $j \in \{0,\dots,\ell\}$ holds. Moreover we say that a call $traverse(X^{[\ell]})$ is *proper* if $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $0 \leq j < \ell$. Let $\ell \in \{0,\dots,k\}$, we shall now demonstrate that a proper call $traverse(X^{[\ell]})$ successfully returns if and only if there exists $w \in L(X^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell] \,[w\rangle_N\, \mathbf{M}_\mathsf{f}[\ell]$.
The formal statement is given at Lem. 4. We give some explanations about Alg. 1 first.

Algorithm 1: *traverse*

Input: A variable $X^{[\ell]} \in \mathcal{X}^{[k]}$ of $G^{[k]}$

1 **begin**
2 Let $p \in \mathcal{P}^{[k]}$ such that $head(p) = X^{[\ell]}$
3 **switch** p **do**
4 **case** $X^{[\ell]} \to \sigma$ /* $\sigma \in \Sigma \cup \{\epsilon\}$ */
5 $\mathbf{M}_\mathsf{i}[\ell] := (\mathbf{M}_\mathsf{i}[\ell] \ominus I(\sigma)) \oplus O(\sigma)$
6 Choose non det $qty \in \mathbb{M}[S]$
7 $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) \ominus qty$
8 **case** $X^{[\ell]} \to B^{[\ell]}C^{[\ell-1]}$
9 $transfer\_from\_to(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{f}[\ell-1])$
10 Choose non det $qty \in \mathbb{M}[S]$
11 $(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) := (\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) \oplus qty$
12 $traverse(C^{[\ell-1]})$
13 assert $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j < \ell$
14 $traverse(B^{[\ell]})$
15 **case** $X^{[\ell]} \to B^{[\ell-1]}C^{[\ell]}$
16 $transfer\_from\_to(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{i}[\ell-1])$
17 Choose non det $qty \in \mathbb{M}[S]$
18 $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) \oplus qty$
19 $traverse(B^{[\ell-1]})$
20 assert $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j < \ell$
21 $traverse(C^{[\ell]})$
22 **return**

Instructions of the form $(var_1, var_2) := (var_1, var_2) \odot qty$ where $qty \in \mathbb{M}[S]$ and $\odot \in \{\oplus, \ominus\}$ stand for the two instructions $var_1 := var_1 \odot qty; var_2 := var_2 \odot qty$. Observe that, given two markings $\mathbf{m}, \mathbf{m}'$, the subtraction operation $\mathbf{m} \ominus \mathbf{m}'$ can be performed only when $\mathbf{m}' \preceq \mathbf{m}$ (i.e., assume every occurence of $\mathbf{m} \ominus \mathbf{m}'$ is preceded by assert $\mathbf{m}' \preceq \mathbf{m}$).

The procedure *transfer_from_to* proceeds as follows: (1) non deterministically choose a marking $qty \in \mathbb{M}[S]$, (2) add qty to tgt, and (3) subtract qty from src. Intuitively, it transfers an arbitrary sub-marking qty of src to tgt.

Algorithm 2: *transfer_from_to*

Input: src, tgt
Choose non det. $qty \in \mathbb{M}[S]$
$tgt := tgt \oplus qty$
$src := src \ominus qty$

Intuitively, $traverse(X^{[\ell]})$ simulates the execution of N along a sequence of transitions w such that $X^{[\ell]} \Rightarrow^* w$. However as *traverse* simulates the derivation of w, it does not necessarily follows a leftmost order but instead an order which guarantees that a bounded amount of memory only is needed to derive w. This is needed for the translation to PNW.

To understand the correctness argument of Alg. 1, let us see why the call $traverse(X^{[\ell]})$ successfully returns if there exists $w \in L(X^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell] \,[w\rangle_N\, \mathbf{M}_\mathsf{f}[\ell]$.

Let us start by assuming that $X^{[\ell]} \Rightarrow u \Rightarrow^* w$ with $u \in (\Sigma \cup \mathcal{X}^{[k]})^*$ and $\mathbf{M}_\mathsf{i}[\ell] \,[w\rangle_N\, \mathbf{M}_\mathsf{f}[\ell]$. An execution of $traverse(X^{[\ell]})$ is such that at line 2, some $p = (X^{[\ell]}, u) \in \mathcal{P}^{[k]}$ is picked. The choice of p yields three case studies.

The first case is given by $p = (X^{[\ell]}, \sigma) \in \mathcal{P}^{[k]}$ (with $\sigma \in \Sigma \cup \{\epsilon\}$) which yields the case of line 4 to be executed. It follows that $X^{[\ell]} \Rightarrow u = w = \sigma$. Since $\mathbf{M}_\mathsf{i}[\ell] \,[\sigma\rangle\, \mathbf{M}_\mathsf{f}[\ell]$ holds by assumption there exists a marking qty such that $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) \ominus qty$

has the effect to empty $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$. Finally, upon reaching line 22 we find that the post condition $\mathbf{M}_\mathsf{i}[\ell] = \varnothing = \mathbf{M}_\mathsf{f}[\ell]$ for every $j \in \{0, \ldots, \ell\}$ holds. Therefore the call to $\mathit{traverse}(X^{[\ell]})$ successfully returns.

The second case is $p = (X^{[\ell]}, B^{[\ell]}C^{[\ell-1]})$ which yields line 9 is executed. We further assume that $\mathbf{M}_\mathsf{i}[\ell] \, [w_1 w_2\rangle \, \mathbf{M}_\mathsf{f}[\ell]$ where $w_1 \in L(B^{[\ell]})$ and $w_2 \in L(C^{[\ell-1]})$. Hence, we find that there is $\mathbf{m} \in \mathbb{M}[S]$ such that $\mathbf{M}_\mathsf{i}[\ell] \, [w_1\rangle \, \mathbf{m} \, [w_2\rangle \, \mathbf{M}_\mathsf{f}[\ell]$ which, by monotonicity of PN, is equivalent to:

$$\exists \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3 \in \mathbb{M}[S] \colon \mathbf{M}_\mathsf{i}[\ell] \, [w_1\rangle \, \mathbf{m}_1 \wedge \mathbf{m}_2 \, [w_2\rangle \, \mathbf{m}_3 \wedge \mathbf{m}_1 = \mathbf{m}_2 \oplus (\mathbf{M}_\mathsf{f}[\ell] \ominus \mathbf{m}_3) \ . \tag{2}$$

Observe that, for (2) to be valid, we need $\mathbf{m}_3 \preceq \mathbf{M}_\mathsf{f}[\ell]$ to hold.

Let us resume the execution of $\mathit{traverse}(X^{[\ell]})$ which now executes the call to the procedure $\mathit{transfer\_from\_to}(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{f}[\ell-1])$ of line 9. We assume the transfer is given by the marking $\mathbf{m}_3$ so that when returning from Alg. 2 we have $\mathbf{M}_\mathsf{f}[\ell-1] = \mathbf{m}_3$. Next the instruction $(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) := (\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) \oplus \mathit{qty}$ of line 11 executes. The effect is to add an arbitrary value, say $\mathbf{m}_2$, to the markings $\mathbf{M}_\mathsf{f}[\ell]$ and $\mathbf{M}_\mathsf{i}[\ell-1]$. Therefore, $\mathbf{M}_\mathsf{i}[\ell-1]$ is updated to $\mathbf{m}_2$ and $\mathbf{M}_\mathsf{f}[\ell]$ to $\mathbf{m}_1$ $(= \mathbf{m}_2 \oplus (\mathbf{M}_\mathsf{f}[\ell] \ominus \mathbf{m}_3))$.

Now, a recursive proper call $\mathit{traverse}(C^{[\ell-1]})$ takes place (see line 12) to determine if there exists a word $w' \in L(C^{[\ell-1]})$ such that $\mathbf{M}_\mathsf{i}[\ell-1] \, [w'\rangle \, \mathbf{M}_\mathsf{f}[\ell-1]$. We conclude from above that w_2 is such a word: $(\mathbf{M}_\mathsf{i}[\ell-1] =)\mathbf{m}_2 \, [w_2\rangle \, \mathbf{m}_3 (= \mathbf{M}_\mathsf{f}[\ell-1])$ holds. Therefore the call $\mathit{traverse}(C^{[\ell-1]})$ successfully returns and we find that $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j < \ell$ (assuming Alg. 1 is correct). This implies that the assert statement at line 13 succeeds.

Then, a recursive proper call $\mathit{traverse}(B^{[\ell]})$ takes place (see line 14) to determine if there exists a word $w' \in L(B^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell] \, [w'\rangle \, \mathbf{M}_\mathsf{f}[\ell]$. We conclude from above that w_1 is such a word: $\mathbf{M}_\mathsf{i}[\ell] \, [w_1\rangle \, \mathbf{m}_1 (= \mathbf{M}_\mathsf{f}[\ell])$ holds. Therefore $\mathit{traverse}(B^{[\ell]})$ successfully returns (again assuming Alg. 1 is correct) and so is $\mathit{traverse}(X^{[\ell]})$ and we are done.

The third case given by $p = (X^{[\ell]}, B^{[\ell-1]}C^{[\ell]}) \in \mathcal{P}^{[k]}$ is treated similarly.

It is worth pointing that the control flow of *traverse* matches the traversal of a parse tree of $G^{[k]}$ such that at each node *traverse* goes first to the subtree which carries the least index. The tree traversal is implemented through recursive calls in *traverse*. To see that the traversal goes first in the subtree of least index, it suffices to look at the ordering of the recursive calls to *traverse* in the code of Alg. 1, e.g. in case the of line 8, $\mathit{traverse}(C^{[\ell-1]})$ is called before $\mathit{traverse}(B^{[\ell]})$. Moreover, we have that the proper call $\mathit{traverse}(X^{[\ell]})$ returns iff there exists a parse tree t of $G^{[k]}$ with root variable $X^{[\ell]}$ such that the sequence of transitions given by the yield of t is enabled from the marking stored in $\mathbf{M}_\mathsf{i}[\ell]$ and its firing yields the marking stored in $\mathbf{M}_\mathsf{f}[\ell]$. Because of the least index first tree traversal, it turns out that the arrays $\mathbf{M}_\mathsf{i}$ and $\mathbf{M}_\mathsf{f}$ provide enough space to manage all the intermediary results.

Also, we observe that when the procedure $\mathit{traverse}(X^{[\ell]})$ calls itself with the parameter, say $B^{[\ell]}$, the call is a *tail recursive call*. This means that when $\mathit{traverse}(B^{[\ell]})$ returns then $\mathit{traverse}(X^{[\ell]})$ immediately returns. It is known from programming techniques how to implement tail recursive call without consuming space on the call stack. In the case of Alg. 1, we can do so by having a global variable to store the parameter of *traverse* and by replacing tail recursive calls with **goto** statements. For the remaining recursive calls (line 12 and 19), because the index of the callee is one less than the index of the caller, we conclude that a bounded space consisting of k frames suffices for the call stack.

Those two insights (two arrays with k entries and a stack with k frames) will be the key to show, in Part 2, that *traverse* can be implemented as a PNW.

▶ **Lemma 4.** *Let* $\ell \in \{0, \ldots, k\}$, $X^{[\ell]} \in \mathcal{X}^{[k]}$, *and* $\mathbf{m}, \mathbf{m}' \in \mathbb{M}[S]$. *Then, the proper call*

traverse$(X^{[\ell]})$ *with* $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$ *and* $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ *successfully returns if and only if there exists* $w \in L(X^{[\ell]})$ *such that* $\mathbf{m}\,[w\rangle_N\,\mathbf{m}'$.

Proof. If. We prove that if there exists $w \in L(X^{[\ell]})$ such that $\mathbf{m}\,[w\rangle\,\mathbf{m}'$ then the proper call *traverse*$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$ and $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ successfully returns.

Our proof is done by induction on the length n of the derivation of $w \in L(X^{[\ell]})$. For the case $n = 1$, we necessarily have $X^{[\ell]} \Rightarrow w = \sigma$ for some $(X^{[\ell]}, \sigma) \in \mathcal{P}^{[k]}$. In this case, the proper call *traverse*$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$ and $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ executes as follows: $p = (X^{[\ell]}, \sigma)$ is picked and the case of line 4 executes successfully since $\mathbf{m} = \mathbf{M}_\mathsf{i}[\ell]\,[\sigma\rangle\,\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ holds. In fact, after the assignment of line 5 we have $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{M}_\mathsf{f}[\ell]$. Hence, by choosing the right *qty*, the instruction $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) \ominus \mathit{qty}$ of line 7 empties $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$ which shows that *traverse*$(X^{[\ell]})$ successfully returns.

For the case $n > 1$, we have $X^{[\ell]} \Rightarrow^n w$ which necessarily has the form $X^{[\ell]} \Rightarrow B^{[\ell]}C^{[\ell-1]} \Rightarrow^{n-1} w$ or $X^{[\ell]} \Rightarrow B^{[\ell-1]}C^{[\ell]} \Rightarrow^{n-1} w$ by def. of $G^{[k]}$. Assume we are in the latter case. Thus there exists w_1 and w_2 such that $X^{[\ell]} \Rightarrow B^{[\ell-1]}C^{[\ell]} \Rightarrow^i w_1C^{[\ell]} \Rightarrow^j w_1w_2 = w$ with $i + j = n - 1$ and $\exists \mathbf{m}_1 \colon \mathbf{m}\,[w_1\rangle\,\mathbf{m}_1\,[w_2\rangle\,\mathbf{m}'$. Observe that $w_1 \in L(B^{[\ell-1]})$ and $w_2 \in L(C^{[\ell]})$ and so by induction hypothesis we find that the proper call *traverse*$(B^{[\ell-1]})$ with $\mathbf{M}_\mathsf{i}[\ell-1] = \mathbf{m}$, $\mathbf{M}_\mathsf{f}[\ell-1] = \mathbf{m}_1$ successfully returns. And so does, by induction hypothesis, the proper call *traverse*$(C^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}_1$, $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$. Therefore let us consider the proper call *traverse*$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$, $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$. We show it successfully returns.

First observe that the call to the procedure *traverse*$(X^{[\ell]})$ is proper. Next, at line 2, pick $p = (X^{[\ell]}, B^{[\ell-1]}C^{[\ell]})$. Then the call *transfer_from_to*$(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{i}[\ell-1])$ of line 16 executes such that $\mathbf{M}_\mathsf{i}[\ell]$ is updated to $\varnothing$ and $\mathbf{M}_\mathsf{i}[\ell-1]$ to $\mathbf{m}$. Next the non determisnistic choice of *qty* and the instruction $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) \oplus \mathit{qty}$ execute such that both $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell-1]$ are updated to $\mathbf{m}_1$. Recall that $\mathbf{m}\,[w_1\rangle\,\mathbf{m}_1\,[w_2\rangle\,\mathbf{m}'$.

Finally we showed above that the proper call *traverse*$(B^{[\ell-1]})$ successfully returns, the assert that follows too and finally the proper call *traverse*$(C^{[\ell]})$. Moreover it is routine to check that upon completion of *traverse*$(C^{[\ell]})$ (and therefore *traverse*$(X^{[\ell]})$) we have $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j \leq \ell$.

The left case (i.e. $p = (X^{[\ell]}, B^{[\ell]}C^{[\ell-1]}) \in \mathcal{P}^{[k]}$) is treated similarly.

Only If. Here we prove that if the proper call *traverse*$(X^{[\ell]})$ successfully returns then there exists $w \in L(X^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell]\,[w\rangle_N\,\mathbf{M}_\mathsf{f}[\ell]$.

Our proof is done by induction on the number n of times line 2 is executed during the execution of *traverse*$(X^{[\ell]})$. In every case, line 2 is executed at least once. For the case $n = 1$, the algorithm necessarily executes the case of line 4. In this case, the definition of $G^{[k]}$ shows that along a successful execution of *traverse*$(X^{[\ell]})$, the non deterministic choice of line 2 necessarily returns a production of the form $p = (X^{[\ell]}, \sigma) \in \mathcal{P}^{[k]}$. Therefore, a successful execution must execute line 5 to 7 and then 22 after which the postcondition $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j \leq \ell$ holds. Because the postcondition holds, we find that $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{M}_\mathsf{f}[\ell]$ holds before executing line 7, hence that $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{M}_\mathsf{i}[\ell] \ominus I(\sigma) \oplus O(\sigma)$ before executing line 5, and finally $\mathbf{M}_\mathsf{i}[\ell]\,[\sigma\rangle\,\mathbf{M}_\mathsf{f}[\ell]$ by semantics of transition σ and we are done.

For the case $n > 1$, the first non deterministic choice of line 2 necessarily picks $p \in \mathcal{P}^{[k]}$ of the form $(X^{[\ell]}, B^{[\ell]}C^{[\ell-1]})$ or $(X^{[\ell]}, B^{[\ell-1]}C^{[\ell]})$. Let us assume $p = (X^{[\ell]}, B^{[\ell]}C^{[\ell-1]})$, hence that the case of line 8 is executed. Let $\mathbf{m}$ and $\mathbf{m}'$ be respectively the values of $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$ when *traverse*$(X^{[\ell]})$ is invoked. Now, let $\mathbf{m}_3, \mathbf{m}_\Delta$ be such that $\mathbf{m}' = \mathbf{m}_3 \oplus \mathbf{m}_\Delta$ and such that upon completion of the call to *transfer_from_to* at line 9 we have that $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}_\Delta$ and $\mathbf{M}_\mathsf{f}[\ell-1] = \mathbf{m}_3$. Moreover, let $\mathbf{m}_2$ be the marking such that $\mathbf{M}_\mathsf{i}[\ell-1] = \mathbf{m}_2$ upon completion of the assignment at line 11. Therefore we find that $\mathbf{M}_\mathsf{f}[\ell]$ is updated to $\mathbf{m}_\Delta \oplus \mathbf{m}_2$. Next consider the successful proper call *traverse*$(C^{[\ell-1]})$ of line 12 with $\mathbf{M}_\mathsf{i}[\ell-1] = \mathbf{m}_2$,

$\mathbf{M}_{\mathsf{f}}[\ell-1] = \mathbf{m}_3$. Observe that because the execution of $traverse(X^{[\ell]})$ yields the calls $traverse(C^{[\ell-1]})$ and $traverse(B^{[\ell]})$, we find that the number of times line 2 is executed in $traverse(C^{[\ell-1]})$ and $traverse(B^{[\ell]})$ is strictly less than n. Therefore, the induction hypothesis shows that there exists w_2 such that $w_2 \in L(C^{[\ell-1]})$ and $\mathbf{m}_2 \,[w_2\rangle\, \mathbf{m}_3$. Then comes the successful assert of line 13 followed by the successful proper call $traverse(B^{[\ell]})$ of line 14 with $\mathbf{M}_{\mathsf{i}}[\ell] = \mathbf{m}$ and $\mathbf{M}_{\mathsf{f}}[\ell] = \mathbf{m}_\Delta \oplus \mathbf{m}_2$. Again by induction hypothesis, there exists w_1 such that $w_1 \in L(B^{[\ell]})$ and $\mathbf{m} \,[w_1\rangle\, (\mathbf{m}_\Delta \oplus \mathbf{m}_2)$.

Next we conclude from the monotonicity property of PN that since $\mathbf{m}_2 \,[w_2\rangle\, \mathbf{m}_3$ then $(\mathbf{m}_2 \oplus \mathbf{m}_\Delta) \,[w_2\rangle\, (\mathbf{m}_3 \oplus \mathbf{m}_\Delta)$, hence that $\mathbf{m} \,[w_1\rangle\, (\mathbf{m}_2 \oplus \mathbf{m}_\Delta) \,[w_2\rangle\, (\mathbf{m}_3 \oplus \mathbf{m}_\Delta)$ and finally that $\mathbf{m} \,[w_1\, w_2\rangle\, \mathbf{m}'$ because $\mathbf{m}' = \mathbf{m}_3 \oplus \mathbf{m}_\Delta$. Finally since $w_1 w_2 \in L(X^{[\ell]})$ we conclude that $\mathbf{m}' \in [\mathbf{m}\rangle^{L(X^{[\ell]})}$ and we are done.

The left case (i.e. $p = (X^{[\ell]}, B^{[\ell-1]}C^{[\ell]}) \in \mathcal{P}^{[k]}$) is treated similarly. ◀

Part 2. In this section, we show that it is possible to construct a PNI N' such that the problem asking if the call to $traverse(A^{[k]})$ successfully returns can be reduced to a reachability problem for N'. Incidentally, we show that N' is a PNW, hence that the reachability problem for PN along finite-index CFL is decidable.

To describe N' we use a generalization of the net program formalism introduced by Esparza in [4] which enrich the instruction set with the test for 0 of a variable.

A *net program* is a finite sequence of *labelled commands*. Those commands have the following form, where $\ell, \ell', \ell_1, \ldots, \ell_k$ are *labels* taken from some arbitrary set, and x is a variable over the natural numbers, also called a *counter*.

ℓ: $x := x - 1$	ℓ: **return**
ℓ: $x := x + 1$	ℓ: **goto** ℓ_1 **or** $\cdots$ **or goto** ℓ_k (where $k \geq 1$)
ℓ: **assert** $x = 0$	ℓ: **gosub** ℓ'

A net program is *syntactically correct* if the labels of commands are pairwise different, and if the destinations of the **goto** and **gosub** commands corresponds to existing labels. (**goto** commands correspond to a possibly non deterministic jump while **gosub** commands correspond to a subroutine call.) A subroutine is a subsequence of the program commands which has a unique *entry label* identified by a *subroutine name*, and a unique *exit command* of the form ℓ: **return**. Also every command of the program belongs to exactly one subroutine. No **goto** commands leaves its enclosing subroutine. Finally, we require the existence of a level assignment to subroutines such that each subroutine only calls lower-level subroutines, which in turn only call lower-level subroutines, etc so as to prevent recursion.

A net program can only be executed once its variables have received initial values which we assume here to be 0. The semantics of net programs can be defined in a straightforward manner from the syntax (see [4] for more information). The only point to be remarked is that the command ℓ: $x := x - 1$ *fails* if $x = 0$, and causes abortion of the program.

The compilation of a syntactically correct net program to a PNI is straightforward and omitted due to space constraints. See [4] for the compilation.

At Alg. 3, 4, 5 and 6 is the net program that implements Alg. 1. In what follows assume S, the set of places of the underlying Petri net, to be $\{1, \ldots, d\}$ for $d \geq 1$. The counter variables of the net program are given by $\{x^{[i]}\}_{0 \leq i \leq k, x \in \mathcal{X}}$ and $\mathbf{M}_{\mathsf{f}}[0..k][1..d]$ $\mathbf{M}_{\mathsf{i}}[0..k][1..d]$ which arranges counters into two matrices of dimension $(k+1) \times d$. For clarity, our net programs use some abbreviations whose semantics is clear from the syntax, e.g. $\mathbf{M}_{\mathsf{i}}[\ell] := \mathbf{M}_{\mathsf{i}}[\ell] \oplus \mathbf{m}$ stands for $\mathbf{M}_{\mathsf{i}}[\ell][1] := \mathbf{M}_{\mathsf{i}}[\ell][1] + \mathbf{m}(1); [\ldots]; \mathbf{M}_{\mathsf{i}}[\ell][d] := \mathbf{M}_{\mathsf{i}}[\ell][d] + \mathbf{m}(d)$.

Let us now make a few observations of Alg. 3, 4, 5 and 6

• the execution starts with the subroutine **main** which sets up $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$, then simulates the call *traverse*$(X^{[\ell]})$ and finally checks that the postcondition holds (label $\mathbf{0}_1$) before returning (label **success**).

• in subroutines $\mathbf{traverse}_j$, [...] stands for the code which is given at Alg. 5 and 6 according to the different cases that may occur. The code for the case $\mathbf{p}_i^j = (X^{[j]}, B^{[j-1]}C^{[j]})$ has been omitted for space reasons but it is easily inferred.

• the counter variables $\{x^{[i]}\}_{0\leq i\leq k, x\in\mathcal{X}}$ record the parameters of the calls to **traverse**. For instance, a call to *traverse*$(X^{[j]})$ is simulated in the net program by incrementing counter $x^{[j]}$ (which records that the parameter of *traverse* is $X^{[j]}$) and then calling subroutine $\mathbf{traverse}_j$. When the call executes, the corresponding variable is decremented.

• the **goto** command at label $\mathbf{traverse}_j$ simulates the non deterministic selection of a production rule $\mathbf{p}_i^j = (X^{[j]}, w)$ which will be fired next (if enabled else the program fails).

Algorithm 3: **main** & $\mathbf{traverse}_{i=\ell,\ldots,0}$

```
main:          M_i[ℓ] := M_i[ℓ] ⊕ m;
               M_f[ℓ] := M_f[ℓ] ⊕ m';
               x^[ℓ] := x^[ℓ] + 1;
               gosub traverse_ℓ;
0_1            assert
               M_i[0..ℓ] = ∅ = M_f[0..ℓ];
success:       return;
traverse_ℓ:    goto p_1^ℓ or ··· or goto p_{n_ℓ}^ℓ;
p_1^ℓ:         [...];
               ⋮
p_{n_ℓ}^ℓ:     [...];
exit^ℓ:        return;
               ⋮
traverse_0:    goto p_1^0 or ··· or goto p_{n_0}^0;
p_1^0:         [...];
               ⋮
p_{n_0}^0:     [...];
exit^0:        return;
```

Algorithm 4: $\mathbf{tr\_f}_\ell\mathbf{\_f}_{\ell-1}$

```
tr_f_ℓ_f_{ℓ-1}:  goto out or t_1 or ... or t_d ;
t_1:             M_f[ℓ][1] := M_f[ℓ][1] − 1;
                 M_f[ℓ−1][1] := M_f[ℓ−1][1]+1;
                 goto tr_f_ℓ_f_{ℓ−1};
                 [...];
t_d:             M_f[ℓ][d] := M_f[ℓ][d] − 1;
                 M_f[ℓ−1][d] := M_f[ℓ−1][d]+1;
                 goto tr_f_ℓ_f_{ℓ−1};
out:             return;
```

Algorithm 5: if $\mathbf{p}_i^j = (X^{[j]}, \sigma)$ then

```
p_i^j:   x^[j] := x^[j] − 1;
         M_i[j] := M_i[j] ⊖ I(σ);
         M_i[j] := M_i[j] ⊕ O(σ);
loop:    goto exit^j or s_1 or ... or s_d ;
s_1:     M_i[j][1] := M_i[j][1] − 1;
         M_f[j][1] := M_f[j][1] − 1;
         goto loop;
         [...];
s_d:     M_i[j][d] := M_i[j][d] − 1;
         M_f[j][d] := M_f[j][d] − 1;
         goto loop;
```

Algorithm 6: if $\mathbf{p}_i^j = (X^{[j]}, B^{[j]}C^{[j-1]})$ then

```
p_i^j:   x^[j] := x^[j] − 1;
         gosub tr_f_j_f_{(j−1)};
loop:    goto exitloop or s_1 or ... or s_d ;
s_1:     M_i[j][1] := M_i[j][1] + 1;
         M_f[j − 1][1] := M_f[j − 1][1] + 1;
         goto loop;
         [...];
s_d:     M_i[j][d] := M_i[j][d] + 1;
         M_f[j − 1][d] := M_f[j − 1][d] + 1;
         goto loop;
exitloop: c^[j−1] := c^[j−1] + 1;
         gosub traverse_{(j−1)};
0_2      assert
         M_i[0..j − 1] = ∅ = M_f[0..j − 1];
l1:      b^[j] := b^[j] + 1;
         goto traverse_j;
```

• the program is syntactically correct. First, observe that no **goto** commands leaves its enclosing subroutine. Second, we assign levels to subroutines as follows: **main** has level $\ell + 1$, $\mathbf{traverse}_j$ has level j for every $0 \leq j \leq \ell$ and $\mathbf{tr\_f}_j\mathbf{\_f}_{j-1}$ has level $j - 1$. Then it is routine to check that this level assignment satisfies the requirement. Moreover, thanks to the programming techniques that allow to implement the tail recursive call as a **goto** instead of **gosub** we find that the program is syntactically correct. (If we had used **gosub** everywhere, then the net program would be syntactically incorrect because of the recursion).

• the **assert** commands at labels $\mathbf{0}_1$ and $\mathbf{0}_2$ have a particular structure matching the level of the subroutines (level $\ell+1$ for $\mathbf{0}_1$ and j for $\mathbf{0}_2$). So, after compilation of the net program into a PNI N', if we set a mapping f from the places of N' to $\mathbb{N}$ such that c is mapped to i if $c \in \{\mathbf{M}_\mathsf{i}[i][j] \mid j \in \{1,\ldots,d\}\} \cup \{\mathbf{M}_\mathsf{f}[i][j] \mid j \in \{1,\ldots,d\}\}$ and every other place is mapped to $\ell+2$ then we find that N' is a PNW. Clearly, deciding whether **main** returns (i.e. reaches **success**) reduces to PNW reachability. Therefore, by Thm. 3, it is decidable whether **main** returns.

▶ **Lemma 5.** *Let $\ell \in \{0,\ldots,k\}$, $X^{[\ell]} \in \mathcal{X}^{[k]}$, and $\mathbf{m},\mathbf{m}' \in \mathbb{M}[S]$. Then the proper call traverse$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$, $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ successfully returns iff* **main** *returns.*

Hence from Lem. 2, 4 and 5, we conclude the following.

▶ **Corollary 6.** *The reachability problem for* PN *along finite-index* CFL *can be reduced to the reachability problem for* PNW*.*

4 From PNW reachability to PN reachability along fiCFL

In this section, we show that the reachability problem for PNW can be reduced to the reachability problem of PN along finite-index CFL. To this aim, let $N = (S,T,F = \langle Z,I,O\rangle,\mathbf{m}_\imath)$ be a PNW, $\mathbf{m}_f \in \mathbb{M}[S]$ a marking, and $f\colon S \to \mathbb{N}$ an index function such that (1) holds.

Let $S = \{s_1,\ldots,s_{n+1}\}$ and $T = \{t_1,\ldots,t_m\}$. Because it simplifies the presentation we will make a few assumptions that yield no loss of generality. (i) For every $i \in \{1,\ldots,n\}$, we have $f(s_i) \leq f(s_{i+1})$, (ii) $\mathbf{m}_\imath = [\![s_{n+1}]\!]$, $\mathbf{m}_f = \varnothing$, (iii) $Z(t_1) \subseteq Z(t_2) \subseteq \cdots \subseteq Z(t_m) \subseteq \{s_1,\ldots,s_n\}$, and (iv) for every $t \in T$, if $s \in Z(t)$ then $O(t)(s) = 0$ (see [15], Lemma 2.1). Notice that the Petri net N can not test if the place s_{n+1} is empty or not.

In the following, we show that it is possible to construct a Petri net (without inhibitor arcs) N', a marking $\mathbf{m}'_f$, and a finite-index CFL L such that: $\mathbf{m}_f \in [\mathbf{m}_\imath\rangle_N^{T^*}$ iff $\mathbf{m}'_f \in [\mathbf{m}'_\imath\rangle_{N'}^{L}$.

Constructing the Petri net N': Let $N' = (S',T',F' = \langle I',O'\rangle,\mathbf{m}'_\imath)$ be a PN which consists in $n+1$ unconnected PN widget: the widget N_0 given by N without tests for zero (i.e. $Z(t)$ is set to $\varnothing$ for every $t \in T$) and the widgets $N_1,\ldots,N_n$ where each $N_i = (\{r_i\},\{p_i,c_i\},F_i,\varnothing)$ where $F_i(p_i) = \langle\varnothing,[\![r_i]\!]\rangle$ and $F_i(c_i) = \langle[\![r_i]\!],\varnothing\rangle$. N_i is depicted as follows: 

. Finally, define $\mathbf{m}'_\imath \in \mathbb{M}[S']$ to be $\mathbf{m}'_\imath(s) = \mathbf{m}_\imath(s)$ for $s \in S$ and 0 elsewhere; and $\mathbf{m}'_f = \varnothing$.

Since we have the ability to restrict the possible sequences of transitions that fire in N', we can enforce the invariant that the sum of tokens in s_i and r_i stays constant. To do so it suffices to force that whenever a token is produced in s_i then a token is consumed from r_i and vice versa. Call L the language enforcing that invariant. Then, let $\mathbf{m}$ be a marking such that $\mathbf{m}(s_i) = \mathbf{m}(r_i) = 0$, observe that by firing from $\mathbf{m}$ a sequence of the form: (i) p_i repeated n times, (ii) any sequence $w \in L$ and (iii) c_i repeated n times; the marking $\mathbf{m}'$ that is reached is such that $\mathbf{m}'(s_i) = \mathbf{m}'(r_i) = 0$. This suggests that to simulate faithfully a transition t_0 of N that does test s_i for 0 we allow the occurrence of the counterpart of t_0 in N_0 right before (i) or right after (iii) only. In what follows, we build upon the above idea the language L_n which, as we we will show, coincides with the finite-index approximation of some CFG.

We need the following notation. Given a word $v \in \Sigma^*$ and $\Theta \subseteq \Sigma$, we define $v|_\Theta$ to be the word obtained from v by erasing all the symbols that are not in Θ. We extend it to languages as follows: Let $L \subseteq \Sigma^*$. Then $L|_\Theta = \{u|_\Theta \mid u \in L\}$.

Constructing the language L_n: For every $j \in \{1,\ldots,m\}$, let $u_j = p_1^{i_1}p_2^{i_2}\cdots p_n^{i_n}$ and $v_j = c_1^{k_1}c_2^{k_2}\cdots c_n^{k_n}$ be two words over the alphabet T' such that $i_\ell = I(t_j)(s_\ell)$ and $k_\ell =$

$O(t_j)(s_\ell)$ for all $\ell \in \{1,\dots,n\}$. Observe that firing the sequence of transitions u_j (resp. v_j) will produce in (resp. consume from) the place r_ℓ (with $1 \leq \ell \leq n$) the same number of tokens that the transition t_j will consume from (resp. produce in) the place s_ℓ. Therefore, firing the sequence of transitions $v_j t_j u_j$ keeps unchanged the total number of tokens in $\{s_i, r_i\}$ for each $i \in \{1,\dots,n\}$.

Let L_0 be a regular language over the alphabet T' defined as follows:

$$L_0 = \{v_j \cdot t_j \cdot u_j \mid Z(t_j) = \emptyset \text{ in the Petri net } N\}^* \ .$$

Next, we define the CFL $L_1,\dots,L_n$ such that for every $\ell \in \{1,\dots,n\}$ we have:

$$L_\ell = \left(\{p_\ell^i \cdot v \cdot c_\ell^i \mid i \in \mathbb{N}, v \in L_{\ell-1}\} \cup \{v_j \cdot t_j \cdot u_j \mid Z(t_j) = \{s_1,\dots,s_\ell\} \text{ in } N\}\right)^* \ .$$

Observe that, for every $j \in \{1,\dots,n\}$, the sum of tokens in the places s_j and r_j is preserved after firing a sequence of transitions in L_ℓ. Hence, the places r_ℓ and s_ℓ are empty after firing a sequence of transitions $w \in L_\ell$ from a marking where these places are empty. Also notice that these places can become non-empty during the execution of w. For instance, if w fires $p_\ell^i \cdot v \cdot c_\ell^i$ which first produces i tokens in r_ℓ, then executes v and finally consumes i tokens from r_ℓ. It is worth pointing that along v transitions which produce and/or consume tokens in s_ℓ can be fired. However, since $v \in L_{\ell-1}$ no transition t such that $s_\ell \in Z(t)$ is allowed, that is no test of s_ℓ for 0 is allowed along v. The language L_ℓ imposes that the place s_ℓ can only be tested for 0 along $v_j \cdot t_j \cdot u_j \in L_\ell \setminus L_{\ell-1}$. The underlying idea is that L_ℓ allows to test s_ℓ for 0 provided the places s_ℓ and r_ℓ (and inductively all the places s_j and r_j for $j \leq \ell$) are empty.

It is routine to check that $L_0 \subseteq L_1 \subseteq \cdots \subseteq L_n$ (since $L_{\ell-1} \subseteq \{p_\ell^i \cdot v \cdot c_\ell^i \mid i \in \mathbb{N}, v \in L_{\ell-1}\}$) and $L_n|_T = T^*$ (since $L_n \supseteq \bigcup_{i=0}^n \{v_j \cdot t_j \cdot u_j \mid Z(t_j) = \{s_1,\dots,s_i\}\}$[3]). Also, L_0 is a regular language and therefore there exists a CFG G_0 and a variable A_0 of G_0 such that $L^{(1)}(A_0) = L_0$. Now, let us assume that for L_i there exists a CFG G_i and a variable A_i such that $L^{(i+1)}(A_i) = L_i$. From the definition of L_{i+1} it is routine to check that there exists a CFG G_{i+1} and a variable A_{i+1} such that $L^{(i+2)}(A_{i+1}) = L_{i+1}$. Finally we find that L_n can be captured by the $n+1$-index approximation of a CFG.

Let us make a few observations about the transitions of N' which were carrying out 0 test in N. In L_ℓ no transition t such that $s_{\ell+1} \in Z(t)$ is allowed, that is no test of place $s_{\ell+1}$ for 0 is allowed along any word of L_ℓ. The language L_ℓ imposes that the place s_ℓ can only be tested for 0 along T_ℓ. The intuition is that L_ℓ allows to test s_ℓ for 0 provided all places s_j and r_j for $j \leq \ell$ are empty.

The relation between the reachability problem for N and the reachability problem for N' along L_n is given by the following lemma (whose proof can be found in [1]):

▶ **Lemma 7.** $\mathbf{m}_f(=\varnothing) \in [\mathbf{m}_\iota\rangle_N$ *if and only if* $\mathbf{m}'_f(=\varnothing) \in [\mathbf{m}'_\iota\rangle_{N'}^{L_n}$.

As an immediate consequence of Lemma 7, we obtain the following result:

▶ **Corollary 8.** *The reachability problem for* PNW *can be reduced, in polynomial time, to the reachability problem for* PN *along finite-index* CFL.

5 Conclusion

In this paper, we have shown that the problem of checking whether the intersection of a finite-index context-free language and a Petri net language is empty is decidable. This result is obtained through a non-trivial reduction to the reachability problem for Petri nets with

[3] Note that if $i = 0$ then $\{s_1,\dots,s_i\} = \emptyset$.

weak inhibitor arcs. On the other hand, we have proved that the reachability problem for Petri nets with weak inhibitor arcs can be reduced, in polynomial time, to the emptiness problem of the language obtained from the intersection of a finite-index context-free language and a Petri net language.

References

1 Mohamed Faouzi Atig and Pierre Ganty. Approximating petri net reachability along context-free traces. *CoRR*, abs/1105.1657, 2011.

2 Rémi Bonnet. The reachability problem for vector addition systems with one zero-test. In *MFCS '11: Proc. 36th Int. Symp. on Mathematical Foundations of Computer Science*, volume 6907 of *LNCS*, pages 145–157. Springer, 2011.

3 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252. ACM Press, 1977.

4 Javier Esparza. Decidability and complexity of petri net problems – an introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998.

5 Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Information Processing Letters*, 111:614–619, 2011.

6 Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newton's method for ω-continuous semirings. In *ICALP '08*, volume 5126 of *LNCS*, pages 14–26. Springer, 2008. Invited paper.

7 Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newtonian program analysis. *Journal of the ACM*, 57(6):33:1–33:47, 2010.

8 Pierre Ganty, Benjamin Monmege, and Rupak Majumdar. Bounded underapproximations. In *CAV '10*, volume 6174 of *LNCS*, pages 600–614. Springer, 2010.

9 Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *CAV '97*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

10 Michel Henri Théodore Hack. Decidability questions for petri nets. Technical Report 161, MIT, 1976.

11 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, third edition, July 2006.

12 Martin Lange and Hans Leiß. To CNF or not to CNF ? An efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8, 2008-2010.

13 Jérôme Leroux. Vector addition system reachability problem (a short self-contained proof). In *POPL '11*, pages 307–316. ACM, 2011.

14 Mark Luker. A family of languages having only finite-index grammars. *Information and Control*, 39(1):14–18, 1978.

15 Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. *Electr. Notes Theor. Comput. Sci*, 223:239 – 264, 2008. RP '08.

16 Arto Salomaa. On the index of a context-free grammar and language. *Information and Control*, 14(5):474 – 477, 1969.

Minimum Fill-in of Sparse Graphs: Kernelization and Approximation*

Fedor V. Fomin[1], Geevarghese Philip[2], and Yngve Villanger[1]

1 **Department of Informatics, University of Bergen, N-5020 Bergen, Norway.** {fomin|yngvev}@ii.uib.no

2 **The Institute of Mathematical Sciences, C.I.T Campus, Taramani, Chennai 600 113, India.** gphilip@imsc.res.in

Abstract

The Minimum Fill-in problem is to decide if a graph can be triangulated by adding at most k edges. The problem has important applications in numerical algebra, in particular in sparse matrix computations. We develop kernelization algorithms for the problem on several classes of sparse graphs. We obtain linear kernels on planar graphs, and kernels of size $\mathcal{O}(k^{3/2})$ in graphs excluding some fixed graph as a minor and in graphs of bounded degeneracy. As a byproduct of our results, we obtain approximation algorithms with approximation ratios $\mathcal{O}(\log k)$ on planar graphs and $\mathcal{O}(\sqrt{k}\log k)$ on H-minor-free graphs. These results significantly improve the previously known kernelization and approximation results for Minimum Fill-in on sparse graphs.

1998 ACM Subject Classification G.2.2 Graph Theory — Graph Algorithms

Keywords and phrases Minimum Fill-In, Approximation, Kernelization, Sparse graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.164

1 Introduction

A graph is *chordal* (or triangulated) if every cycle of length at least four has a chord, i.e. an edge between nonadjacent vertices of the cycle. In the Minimum Fill-in problem (also known as Minimum Triangulation and Chordal Graph Completion) the task is to check if at most k edges can be added to a graph such that the resulting graph is chordal. That is

Minimum Fill-in
Input: A graph $G = (V, E)$ and a non-negative integer k.
Question: Is there $F \subseteq [V]^2$, $\lvert F\rvert \leq k$, such that graph $H = (V, E \cup F)$ is chordal?

This is a classical computational problem motivated by, and named after, a fundamental issue arising in sparse matrix computations. During Gaussian eliminations of large sparse matrices, new non-zero elements — called *fill* — can replace original zeros, thus increasing storage requirements, the time needed for the elimination, and the time needed to solve the system after the elimination. The problem of finding the right elimination ordering

* The research of Fedor V. Fomin was supported by the European Research Council (ERC) grant "Rigorous Theory of Preprocessing", reference 267959. The research of Yngve Villanger was supported by the Research Council of Norway.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 164–175

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

minimizing the amount of fill elements can be expressed as the Minimum Fill-in problem on graphs [21]. Besides sparse matrix computations, applications of Minimum Fill-in can be found in database management, artificial intelligence, and the theory of Bayesian statistics. The survey of Heggernes [15] gives an overview of techniques and applications of minimum and minimal triangulations.

Unfortunately, the problem is notoriously difficult to analyze from the algorithmic perspective. Minimum Fill-in (under the name Chordal Graph Completion) was one of the 12 open problems presented at the end of the first edition of Garey and Johnson's book [13] and it was proved to be NP-complete by Yannakakis [26]. Due to its importance the problem has been studied intensively, and many heuristics, without performance guarantees, have been developed [18, 21].

Very few approximation and FPT algorithms for Minimum Fill-in are known. Chung and Mumford [8] proved that every planar, and more generally, H-minor-free, n-vertex graph has a fill-in with $\mathcal{O}(n \log n)$ edges, thus yielding an $\mathcal{O}(n \log n)$-approximation on these classes of graphs. Agrawal et al. [1] gave an algorithm with the approximation ratio $\mathcal{O}(m^{1.25} \log^{3.5} n/k + \sqrt{m} \log^{3.5} n/k^{0.25})$, where m is the number of edges and n the number of vertices in the input graph. For graphs of degree at most d, they obtained a better approximation factor $\mathcal{O}((nd+k)\sqrt{d} \log^4 n)/k)$. Natanzon et al. [17] provided another type of approximation algorithms for Minimum Fill-in. For an input graph with a minimum fill-in of size k, their algorithm produces a fill-in of size at most $8k^2$, i.e., within a factor of $8k$ of optimal. For graphs with maximum degree d, they gave another approximation algorithm achieving the ratio $\mathcal{O}(d^{2.5} \log^4 (kd))$. Kaplan et al. proved that Minimum Fill-in is fixed parameter tractable (FPT) for the parameter k by giving an algorithm which runs in $\mathcal{O}(k^6 16^k + k^2 mn)$ time [16]. Following this, faster FPT algorithms were devised for the problem, with running times that have smaller constants in the base of the exponent [6, 7]. Very recently, the first and third authors of this paper developed a subexponential FPT algorithm for the problem which runs in $\mathcal{O}(2^{\mathcal{O}(\sqrt{k} \log k)} + k^2 nm)$ time [12].

In this paper we study kernelization algorithms for Minimum Fill-in on different classes of sparse graphs. Kernelization can be regarded as systematic mathematical investigation of preprocessing heuristics within the framework of parameterized complexity. In parameterized complexity each problem instance comes with a parameter k and the parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm (the degree of polynomial is independent of k), called a *kernelization* algorithm, that reduces the input instance down to an instance with size bounded by a polynomial $p(k)$ in k, while preserving the answer. This reduced instance is called a $p(k)$ *kernel* for the problem. If $p(k) = \mathcal{O}(k)$, then we call it a *linear kernel*. For example, for the instance (G, k) of Planar Minimum Fill-in, where G is a planar graph and k is the parameter, the pair (G', k') is a linear kernel if G' is planar, the size of G', i.e., the number of edges and vertices, is $\mathcal{O}(k)$, and there is a fill-in of G with at most k fill edges if and only if there is a fill-in of G' with at most k' fill edges. Kernelization has been extensively studied, resulting in polynomial kernels for a variety of problems. In particular, it has been shown that many problems have polynomial and linear kernels on planar and other classes of sparse graphs [2, 5, 20].

There are several known polynomial kernels for the Minimum Fill-in problem [16] on general (not sparse) graphs. The best known kernelization algorithm is due to Natanzon et al. [17], which for a given instance (G, k) outputs in time $\mathcal{O}(k^2 nm)$ an instance (G', k') such that $k' \leq k$, $|V(G')| \leq 2k^2 + 4k$, and (G, k) is a YES instance if and only if (G', k') is. Note that not every kernelization algorithm for fill-in in general graphs produces a sparse kernel, even if the input is a sparse graph. For example, the algorithm of Natanzon et al. [17],

while reducing the number of vertices in the input graph G, introduces new edges. Thus the resulting kernel G' can be very dense. In order to obtain kernels on classes of sparse graphs, we have to design new kernelization algorithms which preserve the sparsity of the kernel.

Our Results. We provide kernelization algorithms for three important and increasingly general classes of graphs. For planar graphs, we obtain an $\mathcal{O}(k)$ kernel, and for graphs excluding a fixed graph as a minor and graphs of bounded degeneracy, a kernel of size $\mathcal{O}(k^{3/2})$. Our reduction rules are easy to implement. Small kernels for sparse graphs can be used as an argument explaining the successful behavior of several heuristics for sparse matrix computations. As a byproduct of our results, we obtain an approximation algorithm that, for an input planar graph with minimum fill-in of size k, produces a fill-in of size $\mathcal{O}(k \log k)$, which is within factor $\mathcal{O}(\log k)$ of optimal. For H-minor-free graphs our kernelization yields an approximation with the ratio $\mathcal{O}(\sqrt{k} \log k)$.

2 Preliminaries

All graphs in this paper are finite and undirected. In general we follow the graph terminology of Diestel [9]. For a vertex v in graph G, $N_G(v)$ is the set of neighbours of v, and for two non-adjacent vertices u, v, $N_G(u, v) \equiv N_G(u) \cap N_G(v)$. We drop the subscript G where there is no scope for confusion. For $S \subseteq V(G)$, we use $N(S)$ for the set of neighbours in $V(G) \setminus S$ of the vertices in S, and $N[S] \equiv N(S) \cup S$. We also use $G[S]$ to denote the subgraph of G induced by S, and $G \setminus S$ to denote the subgraph $G[V \setminus S]$.

The operation of *contracting* an edge $\{u, v\}$ of a graph consists of replacing its endpoints u, v with a single vertex which is adjacent to all the former neighbours of u and v in G. A graph H is said to be a *contraction* of a graph G if H can be obtained from G by contracting zero or more edges of G. Graph H is a *minor* of G if H is a contraction of some subgraph of G. A family $\mathcal{F}$ of graphs is said to be *H-minor free* if no graph in $\mathcal{F}$ has H as a minor. For $d \in \mathbb{N}$, a graph G is said to be *d-degenerate* if every subgraph of G has a vertex of degree at most d. A family $\mathcal{F}$ of graphs is said to be of *bounded degeneracy* if there is some fixed $d \in \mathbb{N}$ such that every graph in the family is d-degenerate. Note that all graph properties discussed in this paper (being chordal, planar, H-minor free, and d-degenerate) are hereditary, i.e., are closed under taking induced subgraphs.

Minimal Separators. Let u, v be two vertices in a graph G. A set S of vertices of G is said to be a *u, v-separator* of G if u and v are in different components in the graph $G \setminus S$. The set S is said to be a *minimal u, v*-separator if no proper subset of S is a u, v-separator of G. A set S of vertices of G is said to be a *(minimal) separator* of G if there exist two vertices u, v in G such that S is a (minimal) u, v-separator of G.

Let S be a separator of a graph G. A connected component C of $G \setminus S$ is said to be *associated* with S, and is said to be a *full component* if $N(C) = S$.

The following proposition is an exercise in [14].

▶ **Proposition 1.** *A set S of vertices of a graph G is a minimal u, v-separator if and only if u and v are in different full components of $G \setminus S$.*

A set S of vertices of a graph G is said to be a *clique separator* of G if S is a separator of G, and $G[S]$ is a clique.

Minimal and minimum fill-in. *Chordal* or *triangulated* graphs are graphs containing no induced cycles of length more than three. In other words, every cycle of length at least four in a chordal graph contains a chord. Let F be a set of edges which, when added to a graph

G, makes the resulting graph chordal. Then F is called a *fill-in* of G, and the edges in F are called *fill edges*. A fill-in F of G is said to be *minimal* if no proper subset of F is a fill-in of G, and F is a *minimum* fill-in if no fill-in of G contains fewer edges. Notice that every minimum fill-in is also minimal, and so to find a minimum fill-in it is sufficient to search the set of minimal fill-ins.

▶ **Proposition 2.** **[6]** *Let G be a graph, and let S be a minimal separator of G such that $G[S]$ is a complete graph minus one edge, and there is a vertex v in $V(G) \setminus S$ which is adjacent to every vertex in S. Then there exists a minimum fill-in of G which contains the single missing edge in $G[S]$ as a fill-edge.*

The following proposition is folklore; for a proof see, e.g., Bodlaender et al.'s recent article on faster FPT algorithms for the Minimum Fill-In problem [6].

▶ **Proposition 3.** *Let $\langle v_1, v_2, v_3, v_4, \ldots, v_t \rangle$ be a chordless cycle in a graph G, and let F be a minimal fill-in of G. If $\{v_1, v_3\} \notin F$, then $\{v_2, v\} \in F$ for some $v \in \{v_4, \ldots, v_t\}$.*

▶ **Proposition 4.** **[23]** *Let S be a minimal separator of G, let G' be the graph obtained by completing S into a clique, and let $E_S = E(G') \setminus E(G)$. Let $C_1, C_2, \ldots, C_r$ be the connected components of $G \setminus S$. Then $E_S \cup F$ is a minimal fill-in of G if and only if $F = \bigcup_{i=1}^{r} F_i$, where F_i is the set of fill edges in a minimal fill-in of $G'[N[C_i]]$.*

Parameterized complexity. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ. An instance of a parameterized problem is of the form (x, k), where k is called the parameter. A central notion in parameterized complexity is *fixed parameter tractability (FPT)* which means, for a given instance (x, k), solvability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size. We refer to the book of Downey and Fellows [11] for further reading on Parameterized Complexity.

Kernelization. A *kernelization algorithm* for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Gamma^* \times \mathbb{N}$ outputs in time polynomial in $|x| + k$ a pair $(x', k') \in \Gamma^* \times \mathbb{N}$, called the *kernel* such that $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$ and $\max\{k', |x'|\} \leq g(k)$, and $k' \leq k$, where g is some computable function. The function g is referred to as the size of the kernel. If $g(k) = \mathcal{O}(k)$, then we say that Π admits a linear kernel.

The kernels in this paper are obtained by applying a sequence of polynomial time reduction rules. We use the following notational convention: for each reduction rule, (G, k) denotes the instance on which the rule is applied, and (G', k') denotes the resulting instance. We say that a rule is *safe* if (G', k') is a YES instance if and only if (G, k) is a YES instance. We show that each rule is safe. We also show—in most cases—that the resulting graph is in the same class as G.

The remaining part of the paper is organized as follows. Sections 3, 4, and 5 give kernel algorithms for planar, d-degenerate, and H-minor free graphs, respectively. All three kernels use Rule 2 in Section 3, and Rule 6 in Section 4 is used in Section 5 as well. The kernels obtained are then used in Section 6 to get approximations algorithms for planar and H-minor free graphs. We conclude and state some open problems in Section 7.

3 A Linear Kernel for Planar Graphs

In this section we show that the planar minimum fill-in problem has a linear kernel. The kernel is obtained by applying four reduction rules. Rules 1, 2, and 3 are applied exhaustively,

while Rule 4 is only applied if none of the other three can be applied. At the end of this process, the algorithm either solves the problem (giving either YES or NO as the answer), or it yields an equivalent instance $(G', k'); k' \leq k$ where G is of size $\mathcal{O}(k)$.

▶ Reduction Rule 1. **[24]** Let S be a minimal clique separator in G and let $C_1, \ldots, C_t$ be the connected components of $G \setminus S$. We set G' to be the disjoint union of the graphs $G_1, G_2, \ldots, G_t$, where G_i is isomorphic to $G[N[C_i]]$, $1 \leq i \leq t$, and set $k' \leftarrow k$.

By Proposition 4, we have the following lemma.

▶ **Lemma 1.** *Rule 1 is safe.*

Since each of the connected components of graph G' produced by Rule 1 is an induced subgraph of G, it follows that if G is planar or d-degenerate, then G' has the same property. Our next rule deletes vertices which are not part of any chordless cycle; as we show later (Theorem 3), a vertex v satisfies the conditions of the rule if and only if it is not part of any chordless cycle in the graph. This rule can be inferred from previous work due to Tarjan [24] and Berry et al. [3].

▶ Reduction Rule 2. For a vertex v of G, let $C_1, C_2, \ldots, C_t$ be the connected components of $G \setminus N[v]$. If for every $1 \leq i \leq t$, the vertex set $N(C_i)$ is a clique in G, then set $G' \leftarrow G \setminus \{v\}$, $k' \leftarrow k$.

▶ **Lemma 2.** *Rule 2 is safe.*

Proof. Let H be a chordal graph obtained by adding k edges to G. Chordality is a hereditary property, and thus the graph $H' = H \setminus \{v\}$ is chordal. But H' is a triangulation of $G' = G \setminus \{v\}$, and since it is obtained by adding at most k edges, we have that G' has a fill-in of size at most $k' \leq k$.

For the opposite direction, let H' be a *minimal* triangulation obtained from G' by adding the set of fill edges F', where $|F'| \leq k'$. Then the graph H obtained by adding F' to G is chordal. Indeed, if H was not chordal, it would contain a chordless cycle A of length at least 4 passing through v. Let w be a vertex of A not adjacent to v and let C be the connected component of $G \setminus N[v]$ containing w. The set $S = N_G(C)$ is a clique minimal separator in G and thus by Proposition 4, we can conclude that in H every path from w to v should go through some vertex of S. Hence the set S contains at least two non-consecutive (in A) vertices a and b of A. But S is a clique in G, and thus is a clique in H. Hence, a and b form a chord in A, which is a contradiction. Therefore, H is chordal. ◀

In Reduction Rule 2, we only remove a vertex, and thus this rule does not change hereditary properties of graphs, like being H-minor free. We now state some useful properties of graphs on which the above reduction rules cannot be applied.

▶ **Lemma 3.** *A vertex v in a graph G does not satisfy the conditions of Reduction Rule 2 if and only if v is part of a chordless cycle in G.*

Proof. Let v be a vertex in G which does not satisfy the conditions of Reduction Rule 2. Then there exists a connected component C of $G \setminus N[v]$ such that $N(C)$ contains two non adjacent vertices, say $x, y \in N(v)$. Let P be a shortest path from x to y in $G[C \cup \{x, y\}]$. Since x and y are not adjacent, the path P is of length at least two; let $P = \langle x = v_1, v_2, \ldots, v_\ell = y \rangle$. Since P is an induced path, $\langle v, x = v_1, v_2, \ldots, v_\ell = y \rangle$ is a chordless cycle containing v.

Conversely, let $v = v_1, v_2, v_3, \ldots, v_{r-2}, v_{r-1}, v_r = v$ be a chordless cycle in G containing v, and let C be the connected component of $G \setminus N[v]$ which contains v_3 and v_{r-2}. The vertex set $N(C)$ does not contain the edge $\{v_2, v_{r-1}\}$ and hence is not a clique. ◀

▶ **Lemma 4.** *Let G be a graph to which Rule 2 cannot be applied, and let F be an edge set such that $H = (V, E \cup F)$ is chordal. Then for every vertex v in G, there either exists an edge $\{v, x\} \in F$, or an edge $\{u, w\} \in F$, where $u, w \in N(v)$.*

Proof. By Lemma 3 it follows that every vertex v in G is part of at least one chordless cycle $\langle v = v_1, v_2, v_3, v_4, \ldots, v_t \rangle$. By Proposition 3, there is either a fill edge $\{v, v_i\} \in F$ or an edge $\{v_2, v_t\} \in F$, for $i \in \{3, \ldots, t-1\}$. ◀

▶ Reduction Rule 3. **[6]** Let (G, k) be an input instance of Minimum Fill-In. If G has a minimal separator S such that adding exactly one edge to $G[S]$ turns it into a complete graph, and there exists a vertex v in $V(G) \setminus S$ such that all vertices of S are adjacent to v, then

1. Turn $G[S]$ into a complete graph by adding one edge,
2. Apply Rule 1 on the resulting minimal clique separator, and
3. Reduce k by one.

The correctness of this rule is evident from Proposition 2 and Lemma 1. We now show that the rule preserves the planarity of the graph. Observe that if the input graph G is planar, then $|S| \leq 4$.

▶ Claim 1. Reduction Rule 3 preserves the planarity of the graph.

Proof. Let G, S be as in the statement of the rule, and let G' be the graph obtained by applying the rule to G. Let $\{u, v\}$ be the missing edge in $G[S]$. By Proposition 1, there are at least two full components, say C_1, C_2, associated with S in G. Notice that for each $i = 1, 2$, there is a uv-path in G with all internal vertices contained in C_i. This implies that each of the connected components of the output graph G' is a minor of planar graph G, and thus is planar. ◀

▶ Reduction Rule 4. Let (G, k) be an input instance of Minimum Fill-In, where none of the Rules 1, 2, and 3 can be applied. If $|V(G)| > 6k - 4$ then return a trivial NO instance.

▶ **Lemma 5.** *Reduction Rule 4 is safe.*

Proof. Let (G, k) be a YES instance where $G = (V, E)$ is planar and none of the Rules 1, 2, and 3 can be applied. We now argue that $|V| \leq 6k - 4$.

Let F be an edge set such that $|F| \leq k$ and $H = (V, E \cup F)$ is chordal, and let V_F be the set of at most $2k$ vertices that are incident to the edges in F. We then have:

▶ Claim 2. Each vertex $v \in V \setminus V_F$ is adjacent to at least three vertices of V_F.

Proof. Since Rule 2 cannot be applied on vertex v it follows that $N[v] \subsetneq V$. Let C be a connected component of $G \setminus N[v]$ and let $S = N(C)$ be the minimal separator of G separating vertices of C from v. Rules 1 and 3 cannot be applied on S, so the graph $G[S]$ is missing at least two edges $\{x_1, y_1\}$ and $\{x_2, y_2\}$. By finding a shortest path P from x_j to y_j in $G[C \cup \{x_j, y_j\}]$ we can create a chordless cycle consisting of P and x_j, v, y_j for $j \in \{1, 2\}$. By Proposition 3 every fill-in of a chordless cycle either adds an edge incident to vertex v on the chordless cycle or adds a fill edge between its two unique neighbours. By definition there is no fill edge in F incident to v, and thus both $\{x_1, y_1\}$ and $\{x_2, y_2\}$ are contained in F. Two edges have to be incident to at least three vertices, and the claim follows. ◀

We construct a new graph $B = (V, E_B)$ whose edge set E_B is a subset of E, such that $\{u, w\} \in E_B$ if and only if $\{u, w\} \in E$, $u \in V_F$, and $w \notin V_F$. The graph B is planar since it is a subgraph of planar graph G, and is bipartite by construction with the two partite sets being $V_1 = V_F$ and $V_2 = V \setminus V_F$. As noted before, $|V_1| \leq 2k$; we now bound $|V_2|$. Let $\mathcal{F}$ be the set of faces in any fixed planar embedding of B. Let $s = \sum_{f \in \mathcal{F}}(\text{number of edges on the face } f)$. Since B is bipartite, each face has at least four sides, and so $s \geq 4|\mathcal{F}|$. Since each edge of B lies on at most two faces in the embedding, it is counted at most twice in this process, and so $s \leq 2|E_B|$. Thus $4|\mathcal{F}| \leq 2|E_B|$. From this and the well-known Euler's formula for planar graphs applied to B (namely, $|V| - |E_B| + |\mathcal{F}| \geq 2$; observe that B may not be a connected graph) we get $|E_B| \leq 2|V| - 4 = 2(|V_1| + |V_2|) - 4$. By Claim 2 each vertex in V_2 has degree at least 3 in B, and so $E_B \geq 3|V_2|$. Combining these we get $|V_2| \leq 2|V_1| - 4 = 4k - 4$, and so $|V| = |V_1| + |V_2| \leq 6k - 4$. ◀

We now argue that all executions of the rules can be performed in polynomial time. By Proposition 4, a minimal clique separator is a clique separator in every minimal triangulation of the given graph. A minimal triangulation can be constructed in $\mathcal{O}(nm)$ time [22] and the minimal separators of the triangulation which are also cliques in G can be enumerated in $\mathcal{O}(nm)$ time [4]. As a consequence Rule 1 can be executed in polynomial time. For the remaining three rules it is not hard to see that we can check, find an instance, and execute the rule in polynomial time.

The rules are applied exhaustively in the order they are described. Rule 1 is globally applied at most $n - 1$ times, since all minimal clique separators we split on, even across connected components, are the so called "non-crossing" minimal separators in the initial graph, and a graph on n vertices has at most $n - 1$ pairwise non-crossing minimal separators [19]. Each time Rule 1 is applied, at most n connected components are created, and each of them contains at most n vertices. Thus, Rule 2 is applied at most $\mathcal{O}(n^3)$ times. Rule 3 is applied at most k times as one fill edges is added each time, and finally Rule 4 is applied only once. Thus we get

▶ **Theorem 6.** Minimum Fill-In *has a planar kernel of size* $\mathcal{O}(k)$ *in planar graphs.*

4 An $\mathcal{O}(k^{3/2})$ kernel for d-degenerate graphs

We now describe two reduction rules for d-degenerate graphs. The second among these is in fact an algorithm which specifies how to apply Rule 2 and the first rule of this section in tandem. Given a problem instance (G, k) where G is a d-degenerate graph, the second rule outputs an equivalent instance (G', k') such that $k' \leq k$ and $|V(G')| = \mathcal{O}(k^{3/2})$. However, these rules *do not* guarantee that the resulting graph G' is d-degenerate. We will later show how to obtain an equivalent d-degenerate graph from G' while keeping the size bounded by $\mathcal{O}(k^{3/2})$.

The next reduction rule says that if two non-adjacent vertices in an d-degenerate graph G have many common neighbours, then the missing edge between the two vertices belongs to every small fill-in of G.

▶ Reduction Rule 5. Let (G, k) be an instance where G is d-degenerate. Let u, w be two non-adjacent vertices in G, and let $b = |N(u, w)|$. If $(b/2)(b - 1 - 2d) > k$, then set $G' \leftarrow (V(G), E(G) \cup \{\{u, w\}\}), k' \leftarrow k - 1$.

▶ **Lemma 7.** *Rule 5 is safe.*

Algorithm 1 Reduction Rule 6 for d-degenerate graphs.

1: **procedure** RULE6(G, k) ▷ G is assumed to be d-degenerate.
2: **while** (Rule 2 applies to (G, k) and the vertex u) **do**
3: $G \leftarrow G \setminus \{u\}$
4: $F'_0 = \emptyset$
5: **for** (each nonadjacent pair $x, y \in V(G)$) **do**
6: **if** (Rule 5 applies to (G, k) and the non-adjacent vertices x, y) **then**
7: $F'_0 = F'_0 \cup \{\{x, y\}\}$
8: $G' \leftarrow (V(G), E(G) \cup F'_0), k' \leftarrow k - |F'_0|$
9: $D_0 = \emptyset$
10: **while** (Rule 2 applies to (G', k') and the vertex u) **do**
11: $G' \leftarrow G' \setminus \{u\}$,$D_0 = D_0 \cup \{u\}$
12: $F_0 = E(G') \cap F'_0$
13: **if** $k' < 0$ **or** $|V(G')| > 2k + k(2\sqrt{k} + 2d + 1)$ **then**
14: **return** a trivial NO instance.
15: **else**
16: **return** (G', k')

Proof. Let F be a fill-in of G of size at most k. We claim that $\{u, w\} \in F$. For, if $\{u, w\} \notin F$, then let H be the chordal graph obtained by adding the edges in F to the graph G. Since u and w are non-adjacent in H, there exists an u, w-separator in H, and every minimal u, w-separator in H contains all the vertices in $N(u, w)$. Since H is chordal, every minimal separator in H is a clique [10], and so the vertex set $N(u, w)$ induces a clique in H. Hence the subgraph $H[N(u, w)]$ contains $(b - 1)b/2$ edges, where $b = |N(u, w)|$. Since G is d-degenerate, the subgraph $G[N(u, w)]$ contains at most db edges. Thus $|F| \geq (b - 1)b/2 - db = (b/2)(b - 1 - 2d) > k$, a contradiction, and so $\{u, w\} \in F$. It immediately follows that $F \setminus \{\{u, w\}\}$ is a fill-in of G' of size at most $k - 1$.

Conversely, if G' has a fill-in F' of size at most $k - 1$, then $F' \cup \{\{u, w\}\}$ is a fill-in of G of size at most k. ◀

▶ Reduction Rule 6. Let (G, k) be an instance where G is d-degenerate. Set (G', k') to be the instance output by Algorithm 1.

▶ **Lemma 8.** *Rule 6 is safe.*

Proof. By Rule 2 it is safe to delete vertex u in Line 3. Let $e_1, e_2, \ldots, e_{|F'_0|}$ be the set of edges in F'_0. By Rule 5 it is safe to add edge e_1 to G and decrement k. Let our induction hypothesis be that it is safe to add edges $e_1, e_2, \ldots, e_{i-1}$ to G and reduce k by $i-1$, and let us argue that it is also safe to add edges $e_1, e_2, \ldots, e_i$ and reduce k by i. Let $k_{i-1} = k - (i - 1)$. Edge $e_i = \{x, y\}$ was added to F'_0 because $(b/2)(b - 1 - 2d) > k$ where $b = |N_G(x, y)|$. In the extreme case, all the edges $e_1, e_2, \ldots, e_{i-1}$ are added between vertices in $N_G(x, y)$, but $(b/2)(b - 1 - 2d) - (i - 1) > k - (i - 1) = k_{i-1}$ and thus it is safe to add edge e_i as well and reduce k_{i-1} by 1. We can now conclude that (G', k') in Line 8 is a YES instance if and only if (G, k) is. Finally by the safeness of Rule 2, instance (G', k') in Line 13 is a YES instance if and only if (G, k) is.

It remains to argue that we can safely return a trivial NO instance if $|V(G')| > 2k + k(2\sqrt{k} + 2d + 1)$, where G' is the graph in Line 13. Let us assume that (G', k') is a YES instance and let F be a set of edges such that $H = (V(G'), E(G') \cup F)$ is chordal and

$|F| \leq k'$. Let V_F be the set of vertices incident to edges of F, and let V_{F_0} be the set of vertices incident to edges of F_0. Notice that $|V_F| + |V_{F_0}| \leq 2k$ as (G', k') is a YES instance. By Line 10 in Algorithm 1, Rule 2 is applied exhaustively, and thus by Lemma 4 every vertex of $V(G') \setminus (V_F \cup V_{F_0})$ is contained in $N_{G'}(x, y)$ for some edge $\{x, y\} \in F$. In particular, notice that $N_{G'}(x, y) \setminus (V_F \cup V_{F_0}) \subseteq N_G(x, y)$. Since G' is reduced with respect to Rule 5 (See Line 6 of Algorithm 1), $|N_G(x, y)| = b < 2\sqrt{k}+2d+1$. To see this we notice that a clique on b vertices contains $b(b-1)/2$ edges while $G[N_G(x, y)]$ contains at most db edges. Thus if $b \geq 2\sqrt{k}+2d+1$ then $b(b-1)/2-db = b/2(b-1-2d) \geq ((2\sqrt{k}+2d+1)/2)(2\sqrt{k}) > k$ which is a contradiction to $\{x, y\} \notin F'_0$. Summing up we have that $|V(G')| \leq |V_F \cup V_{F_0}| + \sum_{\{x,y\} \in F} |N_G(x, y)| \leq 2k + k(2\sqrt{k} + 2d + 1)$. ◀

Observe that Rule 5 — which is applicable only when the input graph is d-degenerate — adds an edge to the graph. The graph resulting from applying Rule 6 — which adds the edge set F'_0 found by applying Rule 5 — may thus *not* be d-degenerate. The graph output by Rule 6 can be modified to become d-degenerate while preserving the bound on its size, and this gives an $\mathcal{O}(k^{3/2})$ kernel for Minimum Fill-In in d-degenerate graphs:

▶ **Theorem 9.** Minimum Fill-In *has a d-degenerate kernel of size $\mathcal{O}(k^{3/2})$ in d-degenerate graphs.*

Proof. Since a 1-degenerate graph is a forest, and every forest has a fill-in of size zero — since the forest is chordal — we can assume without loss of generality that $d \geq 2$. Let (G, k) be an instance of Minimum Fill-In where G is a d-degenerate graph. The kernelization algorithm applies Reduction Rule 6 – Algorithm 1 – to (G, k) to obtain an equivalent instance (G', k'). If (G', k') is the trivial NO instance returned by Line 14, then it is d-degenerate and its size is a constant, and the kernelization algorithm returns (G', k') itself as the kernel.

Now let (G', k') be a non-trivial instance returned by Line 16. Observe that G' is obtained from G by (i) deleting some vertices – Line 3, – (ii) adding edges F'_0 – Line 6, – and (iii) deleting vertices D_0 – Line 11. Edge set F_0 is defined in Line 12 as the set of edges in F'_0 with both endpoints in $V(G')$.

The kernelization algorithm constructs a new graph G'' from G' by doing the following for each edge $\{u, v\} \in F_0$: remove $\{u, v\}$, add two new vertices a_{uv}, b_{uv}, and make both these vertices adjacent to both u and v. The algorithm returns (G'', k'') as the kernel, where $k'' = k' + |F_0|$. Let G_1 be the graph G' where edge set F_0 is removed.

To see that (G'', k'') satisfies all the requirements, note that G_1 is d-degenerate by the hereditary property of d-degenerate graphs, and $G' = (V(G'), E(G_1) \cup F_0)$. The graph G'' is d-degenerate since it can be obtained from G_1 by adding a sequence of vertices, each of degree two. Since each edge in F_0 corresponds to two new vertices in G'', $|V(G'')| = |V(G_1)| + 2|F_0| \leq 4k + k(2\sqrt{k} + 2d + 1)$.

It remains to argue that (G', k') is a YES instance if and only if (G'', k'') is. If (G', k') is a YES instance, then let F' be a fill-in of G' of size at most k', and let H' be the chordal graph obtained by adding the edges in F' to G'. Let $F'' = F_0 \cup F'$, and let H'' be the graph obtained by adding the edges in F'' to the graph G''. Observe that H'' can be obtained from the chordal graph H' by adding a sequence of vertices of degree two each, each of which is adjacent to the two end-points of some edge in F_0. It follows that H'' is chordal — any potential chordless cycle in H'' has to contain one of these new vertices, but every cycle passing through such a vertex has the respective edge in F_0 as a chord. Thus F'' is a fill-in of G'' of size at most $|F_0| + k' = k''$.

Conversely, let (G'', k'') be a YES instance. Observe that for each $\{u, v\} \in F_0$, the vertex set $S = \{u, v\}$ satisfies all the conditions of Proposition 2 in G'' — S is a minimal

a_{uv}, b_{uv}-separator (using the notation of the proof of Theorem 9), $G''[S]$ is missing the one edge which will make it a clique, and the vertex $a_{uv} \in V(G'') \setminus S$ is adjacent to every vertex in S. So there exists a *minimum* fill-in F'' of G'' such that $F_0 \subseteq F''$, and $|F''| \leq k''$. Let H'' be the chordal graph obtained by adding the edges in F'' to the graph G'', and let H' be the graph obtained by deleting all the vertices $\{a_{uv}, b_{uv} \mid \{u, v\} \in F_0$ from H''. Then H' can be obtained by adding the edges in $F' = F'' \setminus F_0$ to G' , and H' is chordal by the hereditary property of chordality. Thus F' is a fill-in of G' of size at most k'. ◀

5 An $\mathcal{O}(k^{3/2})$ kernel for H-minor free graphs

It is known [25] that every H-minor free graph is d-degenerate for $d \leq \alpha h\sqrt{\log h}$, where $h = |V(H)|$ and $\alpha > 0$ is a constant. As we have already shown in Section 4, the application of Rules 2, 5, and 6 on d-degenerate graphs results in an equivalent instance (G', k') where G' has $\mathcal{O}(k^{3/2})$ vertices. However, this G' is not necessarily H-minor free or d-degenerate. In Theorem 9, we show how to transform G' into a d-degenerate graph without significantly increasing its size. We employ a somewhat more involved transformation to convert G' to an H-minor free problem instance on $\mathcal{O}(k^{3/2})$ vertices:

▶ **Theorem 10.** [⋆][1] *Let H be a fixed graph.* Minimum Fill-In *has an H-minor free kernel of size $\mathcal{O}(k^{3/2})$ in H-minor free graphs.*

6 Approximation

As a byproduct of our kernelization algorithms, we obtain improved approximation algorithms for the Minimum Fill-In problem on planar and H-minor free graphs. We need the following result of Chung and Mumford [8].

▶ **Proposition 5.** [8] *Let H be a fixed graph, and let G be an n vertex graph that is H-minor free. Then there is a triangulation H_T of G such that $|E(H_T)| = \mathcal{O}(n \log n)$, and such a triangulation can be found in polynomial time.*

Together with our improved kernels, this result yields approximate solutions for Minimum Fill-In, with ratio $\mathcal{O}(\sqrt{k} \log k)$ for H-minor-free graphs, and with ratio $\mathcal{O}(\log k)$ for planar graphs.

▶ **Theorem 11.** *Let k be the minimum size of a fill-in of a graph G. There is a polynomial time algorithm which computes a fill-in of G of size $\mathcal{O}(k \log k)$ if G is planar and of size $\mathcal{O}(k^{3/2} \log k)$ if G is H-minor free for some fixed graph H.*

Proof. Let G be a planar graph. For each $k \in \{1, 2, \ldots, n^2\}$, in this order, we run the algorithm of Theorem 6 on (G, k), and compute the maximum value $k^\star$ of the parameter k for which the algorithm gives us a NO answer. This guarantees that there is no fill-in of G of size k^*. We then run the same algorithm on the instance $(G, k^\star + 1)$ to obtain a planar kernel G' on at most $6(k^\star + 1)$ vertices. Using Proposition 5, we obtain a fill-in of G' with at most $c(k^\star + 1) \log (k^\star + 1)$ edges, for some constant c. By making use of standard backtracking, the solution for G' can be transformed into a fill-in of G with $\mathcal{O}(k \log k)$ fill edges.

The arguments when G is an H-minor free graph are almost identical to the planar case. The only difference is that we use Theorem 10 instead, which provides us with an H-minor free kernel of size $\mathcal{O}(k^{3/2})$. ◀

[1] Proofs of results labelled with a ⋆ have been deferred to a longer version of the paper.

7 Conclusion and Open Questions

In this paper we obtained new algorithms for Minimum Fill-In on several sparse classes of graphs. Specifically, we obtained a linear kernel for the problem on planar graphs and kernels of size $\mathcal{O}(k^{3/2})$ in H-minor free graphs and in graphs of bounded degeneracy. Using these kernels, we obtained approximation algorithms with ratios $\mathcal{O}(\log k)$ for planar graphs, and $\mathcal{O}(\sqrt{k}\log k)$ for H-minor free graphs. These results significantly improve known kernelization and approximation results for this problem. We note that for any $g \in \mathbb{N}$, the same set of reduction rules and essentially the same argument as for the planar case shows that Minimum Fill-In has a kernel of size $\mathcal{O}(k)$ in graphs of genus at most g. We conclude with a number of open questions.

Minimum Fill-in on general graphs is NP-complete [26]. However, it is a very old open question if the problem is NP-complete on planar graphs [8]. It turns out that Minimum Fill-in is NP-complete on bipartite 2-degenerate graphs.

▶ **Theorem 12.** [⋆] *The* Minimum Fill-In *problem is NP-complete on bipartite* 2*-degenerate graphs.*

The complexity of the problem on planar and on H-minor free graphs is still open. From the approximation perspective, we leave the possibility of obtaining an $o(\log k)$-approximation on planar graphs as an open problem.

From the perspective of kernelization, it would be very interesting to find out if there is a linear kernel for Minimum Fill-in on H-minor free graphs. We also were not able to find any evidence that the existence of an $\mathcal{O}(k/\log k)$ kernel on planar graphs would contradict any complexity assumption. Can it be that the problem has a sublinear kernel?

Acknowledgements.

We thank our anonymous reviewers for pointing out a way to reduce the constant factor in Reduction Rule 4 from 22 to 6 with a simpler proof of Lemma 5, and for many other comments which helped in improving the presentation. F. V. Fomin acknowledges the support of the European Research Council (ERC) via grant "Rigorous Theory of Preprocessing", reference 267959.

References

1 A. Agrawal, P. N. Klein, and R. Ravi. Cutting down on fill using nested dissection: provably good elimination orderings. *Graph Theory and Sparse Matrix Computation*, 56:31–55, 1993.

2 J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for dominating set. *Journal of the ACM*, 51(3):363–384, 2004.

3 A. Berry, J. P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *Journal of Algorithms*, 58(1):33–66, 2006.

4 J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computations*, pages 1–30. Springer, 1993. IMA Volumes in Mathematics and its Applications, Vol. 56.

5 H. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *FOCS 2009*, pages 629–638. IEEE, 2009.

6 H. Bodlaender, P. Heggernes, and Y. Villanger. Faster parameterized algorithms for minimum fill-in. *Algorithmica*, pages 1–22, 2010. Available online. DOI:10.1007/s00453-010-9421-1.

7 L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.

8 F. R. K. Chung and D. Mumford. Chordal completions of planar graphs. *J. Comb. Theory, Ser. B*, 62(1):96–106, 1994.

9 R. Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, third edition, 2005.

10 G. A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.

11 R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.

12 F. V. Fomin and Y. Villanger. Subexponential parameterized algorithm for minimum fill-in. Accepted at the ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)., 2012.

13 M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

14 M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

15 P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

16 H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28:1906–1922, May 1999.

17 A. Natanzon, R. Shamir, and R. Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM J. Comput.*, 30:1067–1079, October 2000.

18 T. Ohtsuki, L. K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting ordering in a sparse matrix. *J. Math. Anal. Appl.*, 54:622–633, 1976.

19 A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1-3):171–188, 1997.

20 G. Philip, V. Raman, and S. Sikdar. Solving dominating set in larger classes of graphs: Fpt algorithms and polynomial kernels. In *ESA 2009*, volume 5757 of *Lecture Notes in Comput. Sci.*, pages 694–705. Springer, 2009.

21 D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.

22 D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.

23 Kloks. T., D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theor. Comput. Sci.*, 175(2):309–335, 1997.

24 R. E. Tarjan. Decomposition by Clique Separators. *Discrete Mathematics*, 55:221–232, 1985.

25 A. Thomason. The extremal function for complete minors. *Journal of Combinatorial Theory, Series B*, 81(2):318 – 338, 2001.

26 M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

Cubicity, Degeneracy, and Crossing Number

Abhijin Adiga, L. Sunil Chandran, and Rogers Mathew

Department of Computer Science and Automation,
Indian Institute of Science,
Bangalore - 560012, India
{abhijin,sunil,rogers}@csa.iisc.ernet.in

Abstract

A k-box $B = (R_1, R_2, \ldots, R_k)$, where each R_i is a closed interval on the real line, is defined to be the Cartesian product $R_1 \times R_2 \times \cdots \times R_k$. If each R_i is a unit length interval, we call B a k-cube. *Boxicity* of a graph G, denoted as $box(G)$, is the minimum integer k such that G is an intersection graph of k-boxes. Similarly, the *cubicity* of G, denoted as $cub(G)$, is the minimum integer k such that G is an intersection graph of k-cubes.

It was shown in [L. Sunil Chandran, Mathew C. Francis, and Naveen Sivadasan. Representing graphs as the intersection of axis-parallel cubes. *MCDES-2008, IISc Centenary Conference*, available at *CoRR*, abs/cs/0607092, 2006.] that, for a graph G with maximum degree Δ, $cub(G) \leq \lceil 4(\Delta + 1) \ln n \rceil$. In this paper we show that, for a k-degenerate graph G, $cub(G) \leq (k+2)\lceil 2e \log n \rceil$. Since k is at most Δ and can be much lower, this clearly is a stronger result. We also give an efficient deterministic algorithm that runs in $O(n^2k)$ time to output a $8k(\lceil 2.42 \log n \rceil + 1)$ dimensional cube representation for G.

The *crossing number* of a graph G, denoted as $CR(G)$, is the minimum number of crossing pairs of edges, over all drawings of G in the plane. An important consequence of the above result is that if the crossing number of a graph G is t, then $box(G)$ is $O(t^{1/4}\lceil \log t \rceil^{3/4})$. This bound is tight upto a factor of $O((\log t)^{3/4})$.

Let $(\mathcal{P}, \leq)$ be a partially ordered set and let $G_{\mathcal{P}}$ denote its underlying comparability graph. Let $dim(\mathcal{P})$ denote the *poset dimension* of $\mathcal{P}$. Another interesting consequence of our result is to show that $dim(\mathcal{P}) \leq 2(k+2)\lceil 2e \log n \rceil$, where k denotes the degeneracy of $G_{\mathcal{P}}$. Also, we get a deterministic algorithm that runs in $O(n^2k)$ time to construct a $16k(\lceil 2.42 \log n \rceil + 1)$ sized realizer for $\mathcal{P}$. As far as we know, though very good upper bounds exist for poset dimension in terms of maximum degree of its underlying comparability graph, no upper bounds in terms of the degeneracy of the underlying comparability graph is seen in the literature.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Degeneracy, Cubicity, Boxicity, Crossing Number, Interval Graph, Intersection Graph, Poset Dimension, Comparability Graph

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.176

1 Introduction

A graph G is an *intersection graph* of sets from a family of sets $\mathcal{F}$, if there exists $f : V(G) \to \mathcal{F}$ such that $(u, v) \in E(G) \Leftrightarrow f(u) \cap f(v) \neq \emptyset$. Representations of graphs as the intersection graphs of various geometrical objects is a well studied topic in graph theory. Probably the most well studied class of intersection graphs are the *interval graphs*. Interval graphs are the intersection graphs of closed intervals on the real line. A restricted form of interval graphs, that allow only intervals of unit length, are *indifference graphs* or *unit interval graphs*.

An interval on the real line can be generalized to a "k-box" in $\mathbb{R}^k$. A k-box $B = (R_1, R_2, \ldots, R_k)$, where each R_i is a closed interval on the real line, is defined to be the

Cartesian product $R_1 \times R_2 \times \cdots \times R_k$. If each R_i is a unit length interval, we call B a k-cube. Thus, 1-boxes are just closed intervals on the real line whereas 2-boxes are axis-parallel rectangles in the plane. The parameter boxicity of a graph G, denoted as $box(G)$, is the minimum integer k such that G is an intersection graph of k-boxes. Similarly, the cubicity of G, denoted as $cub(G)$, is the minimum integer k such that G is an intersection graph of k-cubes. Thus, interval graphs are the graphs with boxicity equal to 1 and unit interval graphs are the graphs with cubicity equal to 1. A *k-box representation* or a *k dimensional box representation* of a graph G is a mapping of the vertices of G to k-boxes such that two vertices in G are adjacent if and only if their corresponding k-boxes have a non-empty intersection. In a similar way, we define *k-cube representation* (or *k dimensional cube representation*) of a graph G. Since k-cubes by definition are also k-boxes, boxicity of a graph is at most its cubicity.

The concepts of boxicity and cubicity were introduced by F.S. Roberts in 1969 [15]. Roberts showed that for any graph G on n vertices $box(G) \leq \lfloor \frac{n}{2} \rfloor$ and $cub(G) \leq \lfloor \frac{2n}{3} \rfloor$. Both these bounds are tight since $box(K_{2,2,\ldots,2}) = \lfloor \frac{n}{2} \rfloor$ and $cub(K_{3,3,\ldots,3}) = \lfloor \frac{2n}{3} \rfloor$ where $K_{2,2,\ldots,2}$ denotes the complete $n/2$-partite graph with 2 vertices in each part and $K_{3,3,\ldots,3}$ denotes the complete $n/3$-partite graph with 3 vertices in each part. It is easy to see that the boxicity of any graph is at least the boxicity of any induced subgraph of it.

Box representation of graphs finds application in niche overlap (competition) in ecology and to problems of fleet maintenance in operations research (see [9]). Given a low dimensional box representation, some well known NP-hard problems become polynomial time solvable. For instance, the max-clique problem is polynomial time solvable for graphs with boxicity k because the number of maximal cliques in such graphs is only $O((2n)^k)$.

1.1 Previous Results on Boxicity and Cubicity

It was shown by Cozzens [8] that computing the boxicity of a graph is **NP**-hard. Kratochvíl [11] showed that deciding whether the boxicity of a graph is at most 2 itself is **NP**-complete. It has been shown by Yannakakis [19] that deciding whether the cubicity of a given graph is at least 3 is **NP**-hard.

Researchers have tried to bound the boxicity and cubicity of graph classes with special structure. Scheinerman [16] showed that the boxicity of outerplanar graphs is at most 2. Thomassen [17] proved that the boxicity of planar graphs is bounded from above by 3. Upper bounds for the boxicity of many other graph classes such as chordal graphs, AT-free graphs, permutation graphs etc. were shown in [7] by relating the boxicity of a graph with its treewidth. The cube representation of special classes of graphs like hypercubes and complete multipartite graphs were investigated in [15, 12, 13].

Various other upper bounds on boxicity and cubicity in terms of graph parameters such as maximum degree, treewidth etc. can be seen in [4, 2, 3, 10, 7]. The ratio of cubicity to boxicity of any graph on n vertices was shown to be at most $\lceil \log_2 n \rceil$ in [5].

1.2 Equivalent Definitions for Boxicity and Cubicity

Let $G, G_1, G_2, \ldots, G_b$ be a collection of graphs with $V(G) = V(G_i)$, for every $i \leq b$. We say $G = \bigcap_{i=1}^{b} G_i$ when $E(G) = \bigcap_{i=1}^{b} E(G_i)$. Below, we state two very useful lemmas due to Roberts [15].

▶ **Lemma 1.** *For any graph G, $box(G) \leq k$ if and only if there exist k interval graphs $I_1, \ldots, I_k$ such that $G = I_1 \cap \cdots \cap I_k$.*

▶ **Lemma 2.** *For any graph G, $cub(G) \leq k$ if and only if there exist k indifference graphs (unit interval graphs) $I_1, \ldots, I_k$ such that $G = I_1 \cap \cdots \cap I_k$.*

1.3 Our Results

A graph G is *k-degenerate* if the vertices of G can be enumerated in such a way that every vertex is succeeded by at most k of its neighbors. The least number k such that G is k-degenerate is called the degeneracy of G and any such enumeration is referred to as a *degeneracy order* of $V(G)$. For example, trees and forests are 1-degenerate and planar graphs are 5-degenerate. Series-parallel graphs, outerplanar graphs, non-regular cubic graphs, circle graphs of girth at least 5 etc. are subclasses of 2-degenerate graphs.

Main Result: It was shown in [2] that, for a graph G with maximum degree Δ, $cub(G) \leq \lceil 4(\Delta+1)\ln n \rceil$. In this paper, we show that, for a k-degenerate graph G, $cub(G) \leq (k+2)\lceil 2e \log n \rceil$. Since k is at most Δ and can be much lower, this clearly is a stronger result. Moreover, we give an *efficient deterministic algorithm* that outputs a $8k(\lceil 2.42 \log n \rceil + 1)$ dimensional cube representation for G in $O(n^2k)$ time.

Consequence 1: The *crossing number* of a graph G, denoted as $CR(G)$, is the minimum number of crossing pairs of edges, over all drawings of G in the plane. We prove that, if $CR(G) = t$, then $box(G) \leq 66t^{\frac{1}{4}} \lceil \log 4t \rceil^{\frac{3}{4}} + 6$. This bound is tight upto a factor of $O((\log t)^{\frac{3}{4}})$. See Section 5 for details.

Consequence 2: Let $(\mathcal{P}, \leq)$ be a poset (partially ordered set) and let $G_{\mathcal{P}}$ be the underlying comparability graph of $\mathcal{P}$. A linear extension L of $\mathcal{P}$ is a total order which satisfies $(x \leq y \in \mathcal{P}) \implies (x \leq y \in L)$. A realizer of $\mathcal{P}$ is a set of linear extensions of $\mathcal{P}$, say $\mathcal{R}$, which satisfy the following condition: for any two distinct elements x and y, $x \leq y$ in $\mathcal{P}$ if and only if $x \leq y$ in L, $\forall L \in \mathcal{R}$. The *poset dimension* of $\mathcal{P}$, denoted by $dim(\mathcal{P})$, is the minimum integer k such that there exists a realizer of $\mathcal{P}$ of cardinality k. Yannakakis [19] showed that it is NP-complete to decide whether the dimension of a poset is at most 3. The poset dimension is an extensively studied parameter in the theory of partial order (See [18] for a comprehensive treatment).

There are several research papers in the partial order literature which study the dimension of posets whose underlying comparability graph has some special structure – interval order, semi order and crown posets are some examples. While very good upper bounds (for example $c\Delta(\log \Delta)^2$ in [20], where c is a constant) are known for poset dimension in terms of maximum degree Δ of its underlying comparability graph, as far as we know there are no upper bounds in terms of the degeneracy of the underlying comparability graph. Connecting our main result with a result in [1], we can get an upper bound for poset dimension in terms of the degeneracy of the underlying comparability graph as follows. It was shown in [1] that $dim(\mathcal{P}) < 2box(G_{\mathcal{P}})$. Therefore, if the degeneracy of the underlying comparability graph $G_{\mathcal{P}}$ is k, then our result says that $dim(\mathcal{P}) \leq 2(k+2)\lceil 2e \log n \rceil$. Also, we get a deterministic algorithm that runs in $O(n^2k)$ time to construct a $16k(\lceil 2.42 \log n \rceil + 1)$ sized realizer for $\mathcal{P}$.

2 Preliminaries

For any finite positive integer n, let $[n]$ denote the set $\{1, 2, \ldots n\}$. Unless mentioned explicitly, all logarithms are to the base e in this paper. All the graphs that we consider are simple, finite and undirected. For a graph G, we denote the vertex set of G by $V(G)$ and the edge set of G by $E(G)$. For any vertex $u \in V(G)$, $N_G(u) = \{v \in V(G) \mid (u, v) \in E(G)\}$. We define $deg_G(u) := |N_G(u)|$. The average degree of G is denoted by $d_{av}(G)$.

Since an interval graph is the intersection graph of closed intervals on the real line, for every interval graph I_a, there exists a function $f_a : V(I_a) \to \{X \subseteq \mathbb{R} \mid X \text{ is a closed interval}\}$, such that for $u, v \in V(I_a)$, $(u, v) \in E(I_a) \Leftrightarrow f_a(u) \cap f_a(v) \neq \emptyset$. The function f_a is called an *interval representation* of the interval graph I_a. Note that the interval representation of an interval graph need not be unique. Given a closed interval $X = [y, z]$, we define $L(X) := y$ and $R(X) := z$. In a similar way, we call a function f_b a *unit interval representation* of unit interval graph I_b if $f_b : V(I_b) \to \{X' \subseteq \mathbb{R} \mid X' \text{ is a unit length closed interval}\}$, such that $\forall u, v \in V(I_b)$, $(u, v) \in E(I_b) \Leftrightarrow f_b(u) \cap f_b(v) \neq \emptyset$.

Given a graph G, let $\mathcal{C}$ be a coloring of $V(G)$ using colors $\chi_1, \chi_2, \ldots, \chi_a$. Then, for each $u \in V(G)$, $\mathcal{C}(u)$ denotes the color of u in $\mathcal{C}$.

2.1 Definitions, Notations and Assumptions used in Sections 3 and 4:

Recall that the degeneracy of a graph is the least number k such that it has a vertex enumeration in which each vertex is succeeded by at most k of its neighbors. Such an enumeration is called the degeneracy order. The graph G that we consider in these sections is a k-degenerate graph having $V(G) = \{v_1, v_2, \ldots, v_n\}$, $|E(G)| = m$ and $\overline{m}$ $(= \binom{n}{2} - m)$ denotes the number of non-edges in G. The enumeration $v_1, v_2, \ldots, v_n$ is a degeneracy order of $V(G)$ and is denoted by $\mathcal{D}$. For every $v_i, v_j \in V(G)$, we say $v_i <_{\mathcal{D}} v_j$ if v_i comes before v_j in $\mathcal{D}$ i.e., $v_i <_{\mathcal{D}} v_j$ if and only if $i < j$. Suppose $v_i <_{\mathcal{D}} v_j$. If $(v_i, v_j) \in E(G)$, then we call v_j a *forward neighbor* of v_i and v_i is referred to as a *backward neighbor* of v_j. Observe that since G is k-degenerate, a vertex can have at most k forward neighbors. If $(v_i, v_j) \notin E(G)$, then v_j a *forward non-neighbor* of v_i and v_i is a *backward non-neighbor* of v_j. For any $u \in V(G)$, $N^f_G(u) = \{w \in V(G) \mid w \text{ is a forward neighbor of } u\}$ and $N^b_G(u) = \{w \in V(G) \mid w \text{ is a backward neighbor of } u\}$.

Support sets of a non-edge: For each $(v_x, v_y) \notin E(G)$, where $v_x <_{\mathcal{D}} v_y$, let $S_{xy} = \{v_z \in N^f_G(v_x) \mid v_y <_{\mathcal{D}} v_z\} \cup \{v_y\}$. We call S_{xy} the *weak support set* of the non-edge (v_x, v_y). Define $T_{xy} = S_{xy} \cup \{v_x\}$. We call T_{xy} the *strong support set* of the non-edge (v_x, v_y). Let $\mathcal{C}$ be a coloring (need not be proper) of $V(G)$. We say S_{xy} is *favorably colored* in $\mathcal{C}$, if $\mathcal{C}(v_y) \neq \mathcal{C}(v_w)$, $\forall v_w \in S_{xy} \setminus \{v_y\}$. We say T_{xy} is *favorably colored* in $\mathcal{C}$, if $\mathcal{C}(v_y) \neq \mathcal{C}(v_w)$, $\forall v_w \in T_{xy} \setminus \{v_y\}$

3 Cube Representation and Coloring

▶ **Lemma 3.** *Let G be a k-degenerate graph. Let $\chi = \{\chi_1, \chi_2, \ldots \chi_a\}$ be a set of colors and let $\mathbb{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_b\}$ be a family of colorings (need not be proper) of $V(G)$, where each $\mathcal{C}_i$ uses colors from the set χ. If the strong support set T_{xy} of every non-edge $(v_x, v_y) \notin E(G)$, $v_x <_{\mathcal{D}} v_y$, is favorably colored in some $\mathcal{C}_i$, where $i \in [b]$, then $cub(G) \leq ab$.*

Proof. We prove this by constructing ab unit interval graphs $I_{i,j}$ on the vertex set $V(G)$, where $i \in [a]$ and $j \in [b]$, such that $G = \bigcap_{i=1}^{a} \bigcap_{j=1}^{b} I_{i,j}$. Then the statement will follow from Lemma 2. Let $f_{i,j}$ denote an interval representation of $I_{i,j}$. Let us partition the vertices of $I_{i,j}$ into two parts, namely A^{ij} and B^{ij}, where $A^{ij} = \{v \in V(G) \mid \mathcal{C}_i(v) = \chi_j\}$ and $B^{ij} = V(G) \setminus A^{ij}$. For every $i \in [a]$ and $j \in [b]$, an interval representation $f_{i,j}$ of $I_{i,j}$ is constructed from the coloring $\mathcal{C}_i$ in the following way. For every $v_y \in V(G)$,

If $v_y \in A^{ij}$, then

$$f_{i,j}(v_y) = [y + n, y + 2n]$$

else

$$f_{i,j}(v_y) = [g^{ij}_{max}(v_y), g^{ij}_{max}(v_y) + n], \text{ where}$$
$$g^{ij}_{max}(v_y) = \max(\{g \mid (v_y, v_g) \in E(G),$$
$$v_g \in A^{ij}\} \cup \{0\}).$$

Since the length of $f_{i,j}(v_y)$ is n, for every $v_y \in V(G)$, $I_{i,j}$ is a unit interval graph. It is easy to see that, $\forall v_x, v_y \in A^{ij}$, $2n \in f_{i,j}(v_x) \cap f_{i,j}(v_y)$ and therefore A^{ij} forms a clique in $I_{i,j}$. Since $n \in f_{i,j}(v_x) \cap f_{i,j}(v_y)$, $\forall v_x, v_y \in B^{i,j}$, $B^{i,j}$ too forms a clique in $I_{i,j}$. For every $(v_x, v_y) \in E(G)$, with $v_x \in A^{ij}$ and $v_y \in B^{ij}$, we have $L(f_{i,j}(v_y)) = g^{ij}_{max}(v_y) \leq n \leq L(f_{i,j}(v_x)) = n + x \leq n + g^{ij}_{max}(v_y)$, where the last inequality is inferred from the fact that $(v_x, v_y) \in E(G)$ and $v_x \in A^{ij}$. But $n + g^{ij}_{max}(v_y) = R(f_{i,j}(v_y))$. Therefore, we get $L(f_{i,j}(v_y)) \leq L(f_{i,j}(v_x)) \leq R(f_{i,j}(v_y))$ and hence $(v_x, v_y) \in E(I_{i,j})$. Hence $I_{i,j}$ is a supergraph of G.

Let $v_x <_{\mathcal{D}} v_y$ and $(v_x, v_y) \notin E(G)$. We now have to show that there exists some unit interval graph $I_{i,j}$ such that $(v_x, v_y) \notin E(I_{i,j})$. We know that, by assumption, there exists a coloring, say $\mathcal{C}_i$ (where $i \in [a]$), such that the strong support set T_{xy} is favorably colored in $\mathcal{C}_i$. Let $\chi_j = \mathcal{C}_i(v_y)$. Let $g = g^{ij}_{max}(v_x)$. We claim that $g < y$. Assume, for contradiction, that $g > y$. Then $g \neq 0$ and $v_g \in A^{ij}$. Since $y > x$, we get $g > x$. Therefore, $v_g \in N^f_G(v_x)$ and $g > y$. This implies that $v_g \in T_{xy}$. Since T_{xy} is favorably colored in $\mathcal{C}_i$, $\mathcal{C}_i(v_g) \neq \chi_j$. This contradicts the fact that $v_g \in A^{ij}$. Thus we prove the claim. Therefore, $R(f_{i,j}(v_x)) = n + g < n + y = L(f_{i,j}(v_y))$ and hence $(v_x, v_y) \notin E(I_{i,j})$. We infer that $G = \bigcap_{i=1}^{a} \bigcap_{j=1}^{b} I_{i,j}$. ◀

4 Cubicity and Degeneracy

4.1 An Upper Bound – Probabilistic Approach

▶ **Theorem 4.** *For every k-degenerate graph G, $cub(G) \leq (k + 2) \cdot \lceil 2e \log n \rceil$*

Proof. Let $\chi = \{\chi_1, \chi_2, \ldots \chi_{k+2}\}$ be a set of $k + 2$ colors. Generate a random coloring $\mathcal{C}_1$ (need not be a proper coloring) of vertices of G in the following way: For each vertex $v_x \in V(G)$, pick a color χ_j, where $j \in [k + 2]$, uniformly at random from χ and set $\mathcal{C}_1(v_x) = \chi_j$. In a similar way, independently generate random colorings $\mathcal{C}_2, \mathcal{C}_3, \ldots \mathcal{C}_b$, where $b = \lceil 2e \log n \rceil$.

For every $(v_x, v_y) \notin E(G)$ and $v_x <_{\mathcal{D}} v_y$, since G is k-degenerate we have $|T_{xy}| = t \leq k + 2$. $Pr[T_{xy}$ is favorably colored in $\mathcal{C}_i] = \frac{(k+2)(k+1)^{t-1}}{(k+2)^{t-1}} = \left(\frac{k+1}{k+2}\right)^{t-1} \geq \left(\frac{k+1}{k+2}\right)^{k+1}$. Therefore, $Pr[T_{xy}$ is not favorably colored in $\mathcal{C}_i] \leq 1 - \left(\frac{k+1}{k+2}\right)^{k+1} \leq e^{-\left(\frac{k+1}{k+2}\right)^{k+1}}$. Now taking $b = \lceil 2e \log n \rceil$,

$$Pr[\bigcup_{x,y:(v_x <_{\mathcal{D}} v_y),((v_x,v_y) \notin E(G))} \bigcap_{i=1}^{b} (T_{xy} \text{ is not favorably colored in } \mathcal{C}_i)]$$
$$\leq n^2 e^{-b\left(\frac{k+1}{k+2}\right)^{k+1}} < 1.$$

Hence, $Pr[\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_b$ satisfy the condition of Lemma 3$] > 0$. Therefore, there exists a coloring $\mathcal{C}_1, \ldots \mathcal{C}_b$, with $b = \lceil 2e \log n \rceil$, of $V(G)$ using colors from the set $\{\chi_1, \chi_2, \ldots \chi_{k+2}\}$ such that the condition of Lemma 3 is satisfied. Hence by Lemma 3, $cub(G) \leq (k+2) \cdot \lceil 2e \log n \rceil$. ◀

4.2 Deterministic Algorithm

DET_ALGO(G) is a deterministic algorithm which takes a simple, finite k-degenerate graph G as input and outputs a cube representation in $8k\alpha$ dimensional space i.e., $8k\alpha$ unit interval graphs $I_{1,1}, \ldots, I_{1,8k}, \ldots, I_{\alpha,1}, \ldots, I_{\alpha,8k}$ such that $G = \bigcap_{i=1}^{\alpha} \bigcap_{j=1}^{8k} I_{i,j}$. In order to achieve this, DET_ALGO(G) invokes the procedure CONSTRUCT_COLORING (for a detailed version of this procedure , see Appendix A.5) α times and thereby generates α colorings $\mathcal{C}_1, \ldots, \mathcal{C}_\alpha$, where each coloring uses colors from the set $\{\chi_1, \ldots, \chi_{8k}\}$. Then from each coloring $\mathcal{C}_i$, it constructs $8k$ unit interval graphs $I_{i,1}, \ldots, I_{i,8k}$ using the construction described in Lemma 3, which is implemented in procedure CONSTRUCT_UNIT_INTERVAL_GRAPHS (See Appendix A.1).

Note that in order for G to be equal to $\bigcap_{i=1}^{\alpha} \bigcap_{j=1}^{8k} I_{i,j}$, Lemma 3 requires that the colorings $\mathcal{C}_1, \ldots, \mathcal{C}_\alpha$ satisfy the following property: for every $(v_x, v_y) \notin E(G)$, where $v_x <_{\mathcal{D}} v_y$, there exists an $i \in [\alpha]$ such that the strong support set T_{xy} of this non-edge is favorably colored in $\mathcal{C}_i$. The colorings $\mathcal{C}_1, \ldots, \mathcal{C}_\alpha$ are generated one by one keeping this objective in mind. At the stage when we have just generated the $(i-1)$-th coloring $\mathcal{C}_{i-1}$, if a non-edge (v_x, v_y) is such that its strong support set T_{xy} is already favorably colored in some $\mathcal{C}_j$, where $j < i$, then we say that the non-edge (v_x, v_y) is already DONE. Naturally at each stage we have to keep track of the non-edges that are not yet DONE. In order to do this, we introduce two data structures BNN_i and FNN_i, for all $i \in [\alpha]$ [1]. For each $v_y \in V(G)$,

$$\begin{aligned} BNN_i[v_y] &= \{v_x \in V(G) \mid v_x \text{ is a backward non-neighbor of } v_y, \text{ and } (v_x, v_y) \\ &\quad \text{is not yet DONE with respect to } \mathcal{C}_1, \ldots, \mathcal{C}_{i-1}.\} \\ FNN_i[v_y] &= \{v_z \in V(G) \mid v_z \text{ is a forward non-neighbor of } v_y, \text{ and } (v_y, v_z) \\ &\quad \text{is not yet DONE with respect to } \mathcal{C}_1, \ldots, \mathcal{C}_{i-1}.\} \end{aligned}$$

It is easy to see that, $\bigcup_{v_y \in V(G)} BNN_i[v_y] = \bigcup_{v_y \in V(G)} FNN_i[v_y]$ and therefore, $\left(\bigcup_{v_y \in V(G)} BNN_i[v_y] = \emptyset\right) \iff \left(\bigcup_{v_y \in V(G)} FNN_i[v_y] = \emptyset\right)$. In Theorem 7, we show that if we select α to be at least $(\lceil 2.42 \log n \rceil + 1)$, then $FNN_{\alpha+1}[v_y] = \emptyset$, $\forall v_y \in V(G)$. This clearly would mean that all non-edges are DONE with respect to $\mathcal{C}_1, \ldots, \mathcal{C}_\alpha$. In other words, the condition of Lemma 3 will be satisfied for $\mathcal{C}_1, \ldots, \mathcal{C}_\alpha$.

The only thing that remains to be discussed now is how our coloring strategy (i.e. the procedure CONSTRUCT_COLORING) achieves the above objective, namely $BNN_{\alpha+1}[v_y] = \emptyset$ and $FNN_{\alpha+1}[v_y] = \emptyset$, $\forall v_y \in V(G)$, if $\alpha \geq (\lceil 2.42 \log n \rceil + 1)$. To start with $BNN_1[v_y]$ (respectively $FNN_1[v_y]$) contains all the backward (respectively forward) non-neighbors of v_y. The procedure CONSTRUCT_COLORING(i) generates the i-th coloring $\mathcal{C}_i$ as follows. It colors vertices in the reverse degeneracy order starting from vertex v_n. The partial coloring at the stage when we have colored the vertices v_n to v_z is denoted by $\mathcal{C}_i^{v_z}$. Note that $\mathcal{C}_i^{v_1} = \mathcal{C}_i$. Consider the stage at which the algorithm has already colored the vertices from v_n upto v_{y+1} and is about to color v_y. That is, we have the partial coloring $\mathcal{C}_i^{v_{y+1}}$ and are

[1] *BNN* – Backward Non-Neighbor, *FNN* – Forward Non-Neighbor

about to extend it to the partial coloring $\mathcal{C}_i^{v_y}$ by assigning one of the $8k$ possible colors to vertex v_y. Let $\mathcal{C}_i^{v_y=\chi_c}$ denote the partial coloring that results if we extend $\mathcal{C}_i^{v_{y+1}}$ by assigning color χ_c to v_y. The coloring $\mathcal{C}_i$ and the partial colorings $\mathcal{C}_i^{v_z}$, $\forall v_z \in V(G)$ and $\mathcal{C}_i^{v_z=\chi_c}$, $\forall v_z \in V(G), \chi_c \in \{\chi_1, \ldots, \chi_{8k}\}$, will be generically called *the colorings associated with the i-th stage* (i.e. the i-th invocation of CONSTRUCT_COLORING).

With respect to colorings $\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}$ and some coloring $\mathcal{C}'_i$ associated with the i-th stage, we define the following sets:

$$W(v_w, \mathcal{C}'_i) = \{v_x \in BNN_i[v_w] \mid \text{the strong support set } T_{xw} \text{ of non-edge } (v_x, v_w) \text{ is favorably colored in } \mathcal{C}'_i\} \quad (1)$$

$$X(v_w, \mathcal{C}'_i) = \{v_x \in BNN_i[v_w] \mid \text{the weak support set } S_{xw} \text{ of non-edge } (v_x, v_w) \text{ is favorably colored in } \mathcal{C}'_i\} \quad (2)$$

$$Y(v_w, \mathcal{C}'_i) = \{v_z \in FNN_i[v_w] \mid \text{the strong support set } T_{wz} \text{ of non-edge } (v_w, v_z) \text{ is favorably colored in } \mathcal{C}'_i\} \quad (3)$$

$$Z(v_w, \mathcal{C}'_i) = \{v_z \in FNN_i[v_w] \mid \text{the weak support set } S_{wz} \text{ of non-edge } (v_w, v_z) \text{ is favorably colored in } \mathcal{C}'_i\} \quad (4)$$

Naturally, we want to give a color χ_c to v_y such that a large number of (not yet DONE) non-edges incident on v_y get DONE. With respect to the colorings $\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}$ and the partial coloring $\mathcal{C}_i^{v_y=\chi_c}$, we define the status of a non-edge incident on v_y as follows: A non-edge $(v_y, v_z) \in FNN_i[v_y]$ is DONE[2] if T_{yz} is favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$ and is NOT-DONE if T_{yz} is not favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$. A non-edge $(v_x, v_y) \in BNN_i[v_y]$ is HOPELESS[3] if S_{xy} (which happens to be a proper subset of T_{xy}) is not favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$ and is HOPEFUL if S_{xy} is favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$. So when we decide a color for v_y, our intention is to make a large fraction of the HOPEFUL non-edges of $FNN_i[v_y]$ (i.e. the set $Z(v_y, \mathcal{C}_i^{v_y=\chi_c})$), DONE and to make a large fraction of $BNN_i[v_y]$, HOPEFUL. More formally, we want the algorithm to assign a color χ_c to v_y such that the following two conditions are satisfied.
(i) $|X(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|BNN_i[v_y]|$, and
(ii)$|Y(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|Z(v_y, \mathcal{C}_i^{v_y=\chi_c})|$.
The obvious question then is, whether such a color χ_c always exists, for each $v_y \in V(G)$. Lemma 5 answers this question in the affirmative. It follows that, the number of non-edges that are not yet DONE with respect to colorings $\mathcal{C}_1, \ldots \mathcal{C}_i$ is at most a constant fraction of the number of non-edges that were not DONE with respect to colorings $\mathcal{C}_1, \ldots \mathcal{C}_{i-1}$. This is formally proved in Lemma 6. That $BNN_{\alpha+1}[v_y] = \emptyset$ and $FNN_{\alpha+1}[v_y] = \emptyset$, $\forall v_y \in V(G)$, is a consequence of this and is formally proved in Theorem 7.

▶ **Lemma 5.** *For every $i \in [\alpha], v_y \in V(G)$, (i) $|X(v_y, \mathcal{C}_i)| \geq \frac{3}{4}|BNN_i[v_y]|$, and (ii)$|Y(v_y, \mathcal{C}_i)| \geq \frac{3}{4}|Z(v_y, \mathcal{C}_i)|$.*

Proof. See Appendix A.2. ◀

▶ **Lemma 6.** *Let $\overline{m}_i = \Sigma_{y \in [n]}|FNN_i[v_y]|$. Then $\overline{m}_{i+1} \leq \frac{7}{16}\overline{m}_i$.*

---

[2] Recall that we had defined earlier that a non-edge (v_x, v_y) is DONE with respect to a list of colorings $\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}$ if T_{xy} was favorably colored in some C_j, where $j < i$. Here we extend this notion, by allowing the partial coloring $\mathcal{C}_i^{v_y=\chi_c}$ also in the list.

[3] A HOPELESS non-edge (v_x, v_y) will not be DONE with respect to $\mathcal{C}_1, \ldots, \mathcal{C}_i$ if we set $\mathcal{C}_i(v_y) = \chi_c$, irrespective of the color given to $v_{y-1}, \ldots, v_1$.

Algorithm 4.1 DET_ALGO(G)

for $y = n$ to 1 **do**
1. Initialize $BNN_1[v_y] \leftarrow \{v_x \in V(G) \mid v_x <_{\mathcal{D}} v_y, (v_x, v_y) \notin E(G)\}$.
2. Initialize $FNN_1[v_y] \leftarrow \{v_z \in V(G) \mid v_y <_{\mathcal{D}} v_z, (v_y, v_z) \notin E(G)\}$.

end for
3. SET FLAG $\leftarrow$ TRUE.
4. SET i $\leftarrow$ 0.
while FLAG = TRUE **do**
5. i++.
6. $\mathcal{C}_i$ = CONSTRUCT_COLORING(i).
for $y = 1$ to n **do**
7. SET $BNN_{i+1}[v_y] \leftarrow BNN_i[v_y] \setminus W(v_y, \mathcal{C}_i)$
8. SET $FNN_{i+1}[v_y] \leftarrow FNN_i[v_y] \setminus Y(v_y, \mathcal{C}_i)$

end for
9. If $FNN_{i+1}[v_y] = \emptyset$, $\forall v_y \in V(G)$, then FLAG = FALSE.

end while
10. SET $\alpha \leftarrow i$
11. CONSTRUCT_UNIT_INTERVAL_GRAPHS()

Proof. See Appendix A.3. ◀

▶ **Theorem 7.** *Let G be a k-degenerate graph. Algorithm DET_ALGO(G) constructs a valid $8k(\lceil 2.42 \log n \rceil + 1)$ dimensional cube representation for G.*

Proof. The algorithm constructs α colorings $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\alpha$ of $V(G)$, where each coloring uses colors from the set $\{\chi_1, \chi_2, \ldots \chi_{8k}\}$. From Lemma 6, we have $\overline{m}_{i+1} \leq \frac{7}{16}\overline{m}_i$. Also, $\overline{m}_1 = |\Sigma_{y \in [n]} FNN_1[v_y]| \leq n^2$. Putting $\alpha = (\lceil 2.42 \log n \rceil + 1)$, we get $\overline{m}_\alpha \leq 1$. That is, for every $y \in [n]$, $FNN_{\alpha+1}[v_y] = EMPTY$. This means that, for every $(v_x, v_y) \notin E(G)$, where $v_x <_{\mathcal{D}} v_y$, there exists an $i \in [\alpha]$ such that T_{xy} is favorably colored in $\mathcal{C}_i$. Then by Lemma 3 , $cub(G) \leq 8k(\lceil 2.42 \log n \rceil + 1)$. The procedure CONSTRUCT_UNIT_INTERVAL_GRAPHS constructs $8k(\lceil 2.42 \log n \rceil + 1)$ unit interval graphs whose intersection gives G, as described in Lemma 3. Thus we prove the theorem. ◀

4.2.1 Running Time Analysis

▶ **Lemma 8.** *The procedure CONSTRUCT_COLORING(i) can be implemented to run in $O(k\overline{m}_i + kn)$ time, where $\overline{m}_i = \Sigma_{y \in [n]} |FNN_i[v_y]|$.*

Proof. See Appendix A.4. ◀

▶ **Theorem 9.** *DET_ALGO(G) runs in $O(n^2 k)$ time.*

Proof. The algorithm invokes the function CONSTRUCT_COLORING(i) α times to construct colorings $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_\alpha$ of $V(G)$. By Lemma 8, to construct these α colorings it requires $O(\Sigma_{i=1}^{\alpha}(\overline{m}_i k) + \alpha k n)$ time. From Lemma 6, we get that $\Sigma_{i=1}^{\alpha}(\overline{m}_i)$ is $O(\overline{m})$. Since $\alpha = (\lceil 2.42 \log n \rceil + 1)$, the running time of the while loop in DET_ALGO(G) is $O(\overline{m}k + nk \log n)$. It is easy to see that the procedure CONSTRUCT_UNIT_INTERVAL_GRAPHS() runs in $O(nk \log n)$ time. Since $\overline{m} \leq n^2$, DET_ALGO(G) runs in $O(n^2 k)$ time. ◀

Algorithm 4.2 CONSTRUCT_COLORING(i)

/*For a detailed version of this procedure, see Appendix A.5.
All data structures are assumed to be global.
Notational Note:
Let $\mathcal{C}_i^{v_z}$ denote the partial coloring at the stage when we have colored the vertices v_n to v_z. Let $\mathcal{C}_i^{v_z=\chi_c}$ denote the partial coloring that results if we extend $\mathcal{C}_i^{v_{z+1}}$ by assigning color χ_c to v_z.*/

for $y = n$ to 1 **do**
  for each $\chi_c \in \{\chi_1, \ldots, \chi_{8k}\}$ **do**
    1. Compute $|X(v_y, \mathcal{C}_i^{v_y=\chi_c})|$, $|Y(v_y, \mathcal{C}_i^{v_y=\chi_c})|$, and $|Z(v_y, \mathcal{C}_i^{v_y=\chi_c})|$ as per equations (2),(3), and (4) respectively.
    if $|X(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|BNN_i[v_y]|$ and $|Y(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|Z(v_y, \mathcal{C}_i^{v_y=\chi_c})|$ **then**
      2. SET $\mathcal{C}_i^{v_y} \leftarrow \mathcal{C}_i^{v_y=\chi_c}$ (i.e. SET $\mathcal{C}_i(v_y) \leftarrow \chi_c$).
      3. SET $Y(v_y, \mathcal{C}_i^{v_y}) \leftarrow Y(v_y, \mathcal{C}_i^{v_y=\chi_c})$
      4. BREAK.
    end if
  end for
end for
for $y = 1$ to n **do**
  5. Compute $W(v_y, \mathcal{C}_i)$ as per equation (1)
  6. SET $Y(v_y, \mathcal{C}_i) \leftarrow Y(v_y, \mathcal{C}_i^{v_1})$
end for
7. Return $\mathcal{C}_i$.

5 Boxicity and Crossing Number

5.1 A Useful Lemma

For a graph H, let $V_A, V_B \subseteq V(H)$ such that $V(H) = V_A \uplus V_B$. Let $S_B(H)$ be the graph with $V(S_B(H)) = V(H)$ and $E(S_B(H)) = E(H) \setminus \{(u, v) \mid u, v \in V_B\}$. In other words, $S_B(H)$ is obtained from H by making V_B a stable set. Let H_B be the subgraph of H induced on V_B.

▶ **Lemma 10.** *$box(H) \leq 2box(S_B(H)) + box(H_B)$.*

Proof. See Appendix A.6. ◀

5.2 Crossing Number

Crossing number of a graph G, denoted as $CR(G)$, is the minimum number of crossing pairs of edges, over all drawings of G in the plane. A graph G is planar if and only if $CR(G) = 0$. Determination of the crossing number is an NP-complete problem.

The following theorem is due to Pach and Tóth [14]

▶ **Theorem 11.** *For a graph G with n vertices and $m \geq 7.5n$ edges, $CR(G) \geq \frac{1}{33.75}\frac{m^3}{n^2}$, and this estimate is tight upto a constant factor.*

The following claim directly follows from the above theorem.

▶ Claim 12. For a graph G, if $CR(G) \leq t$, then $d_{av}(G) \leq 2(\frac{33.75t}{n})^{1/3} + 15$.

Proof. If $m < 7.5n$, then $d_{av} < 15$. Otherwise, we have $m \leq (33.75n^2t)^{1/3}$ implying that $d_{av} \leq 2(\frac{33.75t}{n})^{1/3}$. ◀

We now prove the main theorem of this section.

▶ **Theorem 13.** *For a graph G with $CR(G) = t$, $box(G) \leq 66 \cdot t^{\frac{1}{4}} \lceil \log 4t \rceil^{\frac{3}{4}} + 6$.*

Proof. Consider a drawing P of G with t crossings. We say a vertex v *participates* in a given crossing in P, if at least one of the edges of the given crossing is incident on v.

Partition the vertices of G into two parts, namely V_A and V_B, such that $V_B = \{v \in V(G) \mid v \text{ participates in some crossing in } P\}$ and $V_A = V(G) \setminus V_B$. Let $S_B(G)$ be the graph with $V(S_B(G)) = V(G)$ and $E(S_B(G)) = E(G) \setminus \{(u, v) \mid u, v \in V_B\}$. In other words, $S_B(G)$ is obtained from G by making V_B a stable set. Let G_B be the subgraph of G induced on V_B. Then by Lemma 10,

$$box(G) \leq 2box(S_B(G)) + box(G_B).$$

Observe that $S_B(G)$ is a planar graph and hence its boxicity is at most 3 (see [17]). Therefore, $box(G) \leq 6 + box(G_B)$. For ease of notation, let $H \equiv G_B$. Then,

$$box(G) \leq 6 + box(H). \tag{5}$$

We have $CR(H) = CR(G) = t$. Let $n = |V(H)|$ and $m = |E(H)|$. At most 4 vertices participate in a given crossing. Since each vertex in H participates in some crossing in P, we get

$$n \leq 4t.$$

Let $V(H) = \{v_1, v_2, \ldots, v_n\}$. Let $v_1, v_2, \ldots, v_n$ be an ordering of the vertices of H, such that for each $i \in [n]$, $deg_{H_i}(v_i) \leq deg_{H_i}(v), \forall v \in V(H_i)$, where H_i denotes the subgraph of H induced on vertex set $\{v_i, v_{i+1}, \ldots, v_n\}$. Let $k = \left(\frac{33.75}{3}\right)^{\frac{1}{4}} \left(\frac{t}{\lceil \log 4t \rceil}\right)^{\frac{1}{4}}$. Let $x = \min(\{i \in [n] \mid deg_{H_i}(v_i) > k\})$. Partition $V(H)$ into two parts, namely $V_C = \{v_1, v_2, \ldots, v_{x-1}\}$ and $V_D = \{v_x, v_{x+1}, \ldots, v_n\}$. Let $S_D(H)$ be the graph with $V(S_D(H)) = V(H)$ and $E(S_D(H)) = E(H) \setminus \{(u, v) \mid u, v \in V_D\}$. In other words, $S_D(H)$ is obtained from H by making V_D a stable set. Let H_D be the subgraph of H induced on V_D. Then by Lemma 10,

$$box(H) \leq 2box(S_D(H)) + box(H_D).$$

Note that $S_D(H)$ is k-degenerate. If $k = 1$, then $S_D(H)$ is a forest and hence its boxicity is at most 2. Suppose $k > 1$. Then by Theorem 4, $box(S_D(H)) \leq cub(S_D(H)) \leq (k + 2)\lceil 2e \log n \rceil \leq 12k\lceil \log(4t) \rceil \leq 12 \left(\frac{33.75}{3}\right)^{\frac{1}{4}} t^{\frac{1}{4}} \lceil \log 4t \rceil^{\frac{3}{4}}$. Thus we have,

$$box(H) \leq 24 \left(\frac{33.75}{3}\right)^{\frac{1}{4}} t^{\frac{1}{4}} \lceil \log 4t \rceil^{\frac{3}{4}} + box(H_D). \tag{6}$$

Since $H_D \equiv H_x$, v_x is a minimum degree vertex of H_D. Therefore, $d_{av}(H_D) > deg_{H_D}(v_x) > k$. Then by Claim 12, we have

$$k = \left(\frac{33.75}{3}\right)^{\frac{1}{4}} \left(\frac{t}{\lceil \log 4t \rceil}\right)^{\frac{1}{4}} < d_{av}(H_D) \leq 2 \left(\frac{33.75t}{|V(H_D)|}\right)^{1/3} + 15.$$

From this, we get $|V(H_D)| \leq 48^{\frac{3}{4}} (33.75t)^{\frac{1}{4}} \lceil \log 4t \rceil^{\frac{3}{4}}$. Since boxicity of a graph is at most half the number of its vertices[15] , we get $box(H_D) \leq \frac{48^{\frac{3}{4}} (33.75t)^{\frac{1}{4}} \lceil \log 4t \rceil^{\frac{3}{4}}}{2}$. Substituting this in Inequality 6, we get

$$box(H) \leq 66t^{\frac{1}{4}} \lceil \log 4t \rceil^{\frac{3}{4}}$$

Therefore from Inequality 5 ,we get

$$box(G) \leq 66t^{\frac{1}{4}}\lceil \log 4t \rceil^{\frac{3}{4}} + 6.$$

◀

5.2.1 Tightness of Theorem 13:

We know that, for any graph G on n vertices and m edges, $CR(G) \leq m(m-1)/2 \leq m^2 \leq n^4$. Let $G \equiv K_{2,2,\ldots,2}$ denote the complete $\frac{n}{2}$-partite graph with 2 vertices in each part and let $t = CR(G)$. From [15], we know that $box(G) = \lfloor \frac{n}{2} \rfloor \geq \lfloor \frac{t^{1/4}}{2} \rfloor$. Therefore, the bound given by Theorem 13 is tight upto a factor of $O((\log t)^{\frac{3}{4}})$.

References

1 Abhijin Adiga, Diptendu Bhowmick, and L. Sunil Chandran. Boxicity and poset dimension. In *COCOON*, pages 3–12, 2010.

2 L. Sunil Chandran, Mathew C. Francis, and Naveen Sivadasan. Representing graphs as the intersection of axis-parallel cubes. *MCDES-2008, IISc Centenary Conference*, available at *CoRR*, abs/cs/0607092, 2006.

3 L. Sunil Chandran, Mathew C. Francis, and Naveen Sivadasan. Boxicity and maximum degree. *Journal of Combinatorial Theory, Series B*, 98(2):443–445, March 2008.

4 L. Sunil Chandran, Mathew C. Francis, and Naveen Sivadasan. Geometric representation of graphs in low dimension using axis parallel boxes. *Algorithmica*, 56(2):129–140, 2010.

5 L. Sunil Chandran and K. Ashik Mathew. An upper bound for cubicity in terms of boxicity. *Discrete Mathematics*, In Press, Corrected Proof, doi:10.1016/j.disc.2008.04.011, 2008.

6 L. Sunil Chandran, Rogers Mathew, and Naveen Sivadasan. Boxicity of line graphs. *CoRR*, abs/1009.4471, 2010.

7 L. Sunil Chandran and Naveen Sivadasan. Boxicity and treewidth. *Journal of Combinatorial Theory, Series B*, 97(5):733–744, September 2007.

8 M. B. Cozzens. Higher and multidimensional analogues of interval graphs. Ph. D. thesis, Rutgers University, New Brunswick, NJ, 1981.

9 M. B. Cozzens and F. S. Roberts. Computing the boxicity of a graph by covering its complement by cointerval graphs. *Discrete Applied Mathematics*, 6:217–228, 1983.

10 Louis Esperet. Boxicity of graphs with bounded degree. *European Journal of Combinatorics*, doi:10.1016/j.ejc.2008.10.003, 2008.

11 J. Kratochvil. A special planar satisfiability problem and a consequence of its NP–completeness. *Discrete Applied Mathematics*, 52:233–252, 1994.

12 H. Maehara. Sphericity exceeds cubicity for almost all complete bipartite graphs. *Journal of Combinatorial Theory, Series B*, 40(2):231–235, April 1986.

13 T.S. Michael and Thomas Quint. Sphericity, cubicity, and edge clique covers of graphs. *Discrete Applied Mathematics*, 154(8):1309–1313, May 2006.

14 János Pach and Géza Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997.

15 F. S. Roberts. *Recent Progresses in Combinatorics*, chapter On the boxicity and cubicity of a graph, pages 301–310. Academic Press, New York, 1969.

16 E. R. Scheinerman. Intersection classes and multiple intersection parameters. Ph. D. thesis, Princeton University, 1984.

17 C. Thomassen. Interval representations of planar graphs. *Journal of Combinatorial Theory, Series B*, 40:9–20, 1986.

18 W.T. Trotter. *Combinatorics and partially ordered sets: Dimension theory.* Johns Hopkins Univ Pr, 2001.
19 Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3:351–358, 1982.
20 Z. Füredi and J. Kahn. On the dimensions of ordered sets of bounded degree. *Order*, 3(1):15–20, 1986.

A Appendix

A.1 Procedure CONSTRUCT_UNIT_INTERVAL_GRAPHS()

Algorithm A.1 CONSTRUCT_UNIT_INTERVAL_GRAPHS()

/*All data structures are assumed to be global. */

1. INITIALIZE $L(f_{i,j}(v_y)) \leftarrow 0, R(f_{i,j}(v_y)) \leftarrow n, \forall y \in [n], i \in \alpha, j \in [8k]$
for $i = 1$ to α **do**
  for $y = n$ to 1 **do**
    2. SET $j \leftarrow c$, such that $\mathcal{C}_i(v_y) = \chi_c$
    3. SET $L(f_{i,j}(v_y)) \leftarrow y + n$
    4. SET $R(f_{i,j}(v_y)) \leftarrow y + 2n$
    for each $v \in N^b_G(v_y)$ **do**
      if $(\mathcal{C}_i(v) \neq j) \cap (L(f_{i,j}(v)) = 0)$ **then**
        5. SET $L(f_{i,j}(v)) \leftarrow y$
        6. SET $R(f_{i,j}(v)) \leftarrow y + n$
      end if
    end for
  end for
end for
7. Output $f_{i,j}(v_y), \forall y \in [n], i \in \alpha, j \in [8k]$

A.2 The proof of Lemma 5

Proof. The statement of the lemma is obvious if the BREAK statement in Step 4 of CONSTRUCT_COLORING(i) is executed, for every $i \in [\alpha]$ and $v_y \in V(G)$. In order to prove that the BREAK statement will be executed, it is sufficient to show that there exists a color $\chi_c \in \{\chi_1, \ldots, \chi_{8k}\}$ such that $|X(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|BNN_i[v_y]|$ and $|Y(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|Z(v_y, \mathcal{C}_i^{v_y=\chi_c})|$. Since the vertices in $Z(v_y, \mathcal{C}_i^{v_y=\chi_c})$ or $Z(v_y, \mathcal{C}_i)$ do not depend on the colors given to $v_1, \ldots v_y$, we have $Z(v_y, \mathcal{C}_i^{v_y=\chi_c}) = Z(v_y, \mathcal{C}_i)$. Hence, $Z(v_y, \mathcal{C}_i^{v_y=\chi_c})$ and $Z(v_y, \mathcal{C}_i)$ can be used interchangeably.

Let $A = BNN_i[v_y] \times Z(v_y, \mathcal{C}_i)$. Let $< v_x, v_z >$ be an element of A. We say a color χ_c is *good* for $< v_x, v_z >$, if $v_x \in X(v_y, \mathcal{C}_i^{v_y=\chi_c})$ and $v_z \in Y(v_y, \mathcal{C}_i^{v_y=\chi_c})$. In other words, χ_c is *good* for $< v_x, v_z >$, if both S_{xy} and T_{yz} are favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$. S_{xy} is favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$, if $\chi_c \notin P$, where $P = \{\mathcal{C}_i^{v_y=\chi_c}(v_w) \mid v_w \in N^f_G(v_x), v_y <_{\mathcal{D}} v_w\}$. Since $|N^f_G(v_x)| \leq k$, $|P| \leq k$. Therefore, there are at least $8k - k = 7k$ possible values that χ_c can take such that S_{xy} is favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$. For T_{yz} also to be favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$, the only thing required is that $\chi_c \neq \mathcal{C}_i^{v_y=\chi_c}(v_z)$, since $v_z \in Z(v_y, \mathcal{C}_i)$ and therefore S_{yz} is already favorably colored. This implies that there are at least $7k-1$ possible values that χ_c can take

such that both S_{xy} and T_{yz} are favorably colored in $\mathcal{C}_i^{v_y=\chi_c}$. In other words, there are at least $7k-1$ *good* colors for $< v_x, v_z >$. Thus for each element in A, there are at least $7k-1$ colors *good* for it. For each color $\chi_j \in \{\chi_1, \ldots, \chi_{8k}\}$, let $S^j = \{< v_x, v_z > \in A \mid \chi_j \text{ is } good \text{ for } < v_x, v_z >\} = X(v_y, \mathcal{C}_i^{v_y=\chi_j}) \times Y(v_y, \mathcal{C}_i^{v_y=\chi_j})$. Since there are at least $(7k-1)$ colors *good* for each element in A, $\Sigma_{j\in[8k]}|S^j| \geq (7k-1)|A|$. Then by pigeonhole principle, there exists a $c \in [8k]$ such that $|S^c| = |X(v_y, \mathcal{C}_i^{v_y=\chi_c})| \cdot |Y(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{(7k-1)}{8k}|A| = \frac{7k-1}{8k}|BNN_i[v_y]| \cdot |Z(v_y, \mathcal{C}_i)| \geq \frac{3}{4}|BNN_i[v_y]| \cdot |Z(v_y, \mathcal{C}_i)|$ elements of A. In other words, $|X(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|BNN_i[v_y]|$ and $|Y(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|Z(v_y, \mathcal{C}_i^{v_y=\chi_c})|$. ◀

A.3 The proof of Lemma 6

Proof. From step 8 of DET_ALGO(G), we have $|FNN_{i+1}[v_y]| = |FNN_i[v_y]| - |Y(v_y, \mathcal{C}_i)| \leq |FNN_i[v_y]| - \frac{3}{4}|Z(v_y, \mathcal{C}_i)|$ (using Lemma 5). Taking summation over all $y \in [n]$, we get $\overline{m}_{i+1} \leq \overline{m}_i - \frac{3}{4}\Sigma_{y\in[n]}|Z(v_y, \mathcal{C}_i)| = \overline{m}_i - \frac{3}{4}\Sigma_{y\in[n]}|X(v_y, \mathcal{C}_i)|$. The last equality comes from the fact that both $\Sigma_{y\in[n]}|X(v_y, \mathcal{C}_i)|$ and $\Sigma_{y\in[n]}|Z(v_y, \mathcal{C}_i)|$ represent the number of HOPEFUL non-edges in G with respect to colorings $\mathcal{C}_1, \ldots, \mathcal{C}_i$. From Lemma 5, we have $|X(v_y, \mathcal{C}_i)| \geq \frac{3}{4}|BNN_i[v_y]|$. Therefore, $\overline{m}_{i+1} \leq \overline{m}_i - (\frac{3}{4})^2\Sigma_{y\in[n]}|BNN_i[v_y]|$. Since $\Sigma_{y\in[n]}|BNN_i[v_y]| = \Sigma_{y\in[n]}|FNN_i[v_y]|$, we get $\overline{m}_{i+1} \leq \overline{m}_i - (\frac{3}{4})^2\Sigma_{y\in[n]}|FNN_i[v_y]| = \overline{m}_i - \frac{9}{16}\overline{m}_i = \frac{7}{16}\overline{m}_i$. ◀

A.4 The proof of Lemma 8

Proof. A detailed description of the procedure is given in Section A.5. To implement the procedure efficiently, we make use of an $(n \times 8k)$ $0-1$ matrix, hereafter called *FNC* (Forward Neighbor Color), and two $(n \times n)$ $0-1$ matrices named *HOPE_MATRIX* and *DONE_MATRIX* respectively. At the beginning of the procedure each of these matrices have all entries set to 0. As the procedure progresses, we change some of the entries to 1 in such a way that,
$\forall w \in [n], j \in [8k], FNC[w][j] = 1 \iff \exists v_z \in N_G^f(v_w)$ such that v_z is already colored by the procedure with color χ_j.
$\forall w, z \in [n], v_w \in BNN_i[v_z], HOPE\_MATRIX[w][z] = 1 \iff S_{wz}$ is already favorably colored by the procedure.
$\forall w, z \in [n], v_w \in BNN_i[v_z], DONE\_MATRIX[w][z] = 1 \iff T_{wz}$ is already favorably colored by the procedure.

In order for the above matrices to satisfy their respective properties, the only thing that needs to be done is to update these matrices at each stage of the procedure. Consider the stage at which the procedure is extending partial coloring $\mathcal{C}_i^{v_y+1}$ to $\mathcal{C}_i^{v_y}$ by assigning color χ_c to v_y. At this stage, the matrices *FNC*, *HOPE_MATRIX* and *DONE_MATRIX* are updated as described in steps $11(a)$, $12(a)$ and $13(a)$ respectively. Note that this can be done in $O(|BNN_i[v_y]| + |FNN_i[v_y]| + |N_G^b(v_y)|)$ time. Steps 4(a)-(b), 5(a)-(b) and 6(a)-(b) compute $X(v_y, \mathcal{C}_i^{v_y=\chi_c}), Y(v_y, \mathcal{C}_i^{v_y=\chi_c})$ and $Z(v_y, \mathcal{C}_i^{v_y=\chi_c})$ respectively in $O(|BNN_i[v_y]| + |FNN_i[v_y]|)$ time. Computing $W(v_y, \mathcal{C}_i)$ is done in step 15 (a)–(b) in $O(|BNN_i[v_y]|)$ time.

Since steps 4 to 14, in the worst case, are run for each $v_y \in V(G), \chi_c \in \{\chi_1, \ldots, \chi_{8k}\}$, the procedure runs in $O(k(\Sigma_{y\in[n]}(|BNN_i[v_y]| + |FNN_i[v_y]|) + \Sigma_{y\in[n]}|N_G^b(v_y)|))$ time. We know that $\Sigma_{y\in[n]}(|BNN_i[v_y]| + |FNN_i[v_y]|) = 2\overline{m}_i$ and $\Sigma_{y\in[n]}|N_G^b(v_y)| = m \leq kn$. Hence the Lemma. ◀

A.5 A Detailed version of procedure CONSTRUCT_COLORING(i)

Algorithm A.2 CONSTRUCT_COLORING(i) /* detailed */

/*All data structures are assumed to be global.
Notational Note:
Let $\mathcal{C}_i^{v_z}$ denote the partial coloring at the stage when we have colored the vertices v_n to v_z. Let $\mathcal{C}_i^{v_z=\chi_c}$ denote the partial coloring that results if we extend $\mathcal{C}_i^{v_{z+1}}$ by assigning color χ_c to v_z. */

1. Initialize $FNC[w][j] \leftarrow 0, \forall w \in [n], j \in [8k]$
2. Initialize $HOPE\_MATRIX[w][z] \leftarrow 0, \forall w, z \in [n]$
3. Initialize $DONE\_MATRIX[w][z] \leftarrow 0, \forall w, z \in [n]$

for $y = n$ to 1 **do**
  for each $\chi_c \in \{\chi_1, \ldots, \chi_{8k}\}$ **do**
    4. Compute $X(v_y, \mathcal{C}_i^{v_y=\chi_c})$ /*as described in steps (a) and (b) below */
       (a) Initialize $X(v_y, \mathcal{C}_i^{v_y=\chi_c}) \leftarrow \emptyset$
       (b) $\forall v_x \in BNN_i[v_y]$, if $FNC[x][c] = 0$, then
       SET $X(v_y, \mathcal{C}_i^{v_y=\chi_c}) \leftarrow X(v_y, \mathcal{C}_i^{v_y=\chi_c}) \cup \{v_x\}$
    5. Compute $Y(v_y, \mathcal{C}_i^{v_y=\chi_c})$ /*as described in steps (a) and (b) below */
       (a) Initialize $Y(v_y, \mathcal{C}_i^{v_y=\chi_c}) \leftarrow \emptyset$
       (b) $\forall v_z \in FNN_i[v_y]$, if $(HOPE\_MATRIX[y][z] = 1)$ and
       $\left(\mathcal{C}_i^{v_y=\chi_c}(v_z) \neq \chi_c\right)$, then SET $Y(v_y, \mathcal{C}_i^{v_y=\chi_c}) \leftarrow Y(v_y, \mathcal{C}_i^{v_y=\chi_c}) \cup \{v_z\}$
    6. Compute $Z(v_y, \mathcal{C}_i^{v_y=\chi_c})$ /*as described in steps (a) and (b) below */
       (a) Initialize $Z(v_y, \mathcal{C}_i^{v_y=\chi_c}) \leftarrow \emptyset$
       (b) $\forall v_z \in FNN_i[v_y]$, if $HOPE\_MATRIX[y][z] = 1$,
       then SET $Z(v_y, \mathcal{C}_i^{v_y=\chi_c}) \leftarrow Z(v_y, \mathcal{C}_i^{v_y=\chi_c}) \cup \{v_z\}$
    if $|X(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|BNN_i[v_y]|$ and $|Y(v_y, \mathcal{C}_i^{v_y=\chi_c})| \geq \frac{3}{4}|Z(v_y, \mathcal{C}_i^{v_y=\chi_c})|$ **then**
      7. SET $\mathcal{C}_i^{v_y} \leftarrow \mathcal{C}_i^{v_y=\chi_c}$ (i.e. SET $\mathcal{C}_i(v_y) \leftarrow \chi_c$).
      8. SET $X(v_y, \mathcal{C}_i^{v_y}) \leftarrow X(v_y, \mathcal{C}_i^{v_y=\chi_c})$
      9. SET $Y(v_y, \mathcal{C}_i^{v_y}) \leftarrow Y(v_y, \mathcal{C}_i^{v_y=\chi_c})$
      10. SET $Z(v_y, \mathcal{C}_i^{v_y}) \leftarrow Z(v_y, \mathcal{C}_i^{v_y=\chi_c})$
      11. Update FNC matrix. /* as described in step (a) below */
          (a) $\forall v_x \in N_G^b(v_y)$, SET $FNC[x][c] \leftarrow 1$
      12. Update $HOPE\_MATRIX$ /* as described in step (a) below */
          (a) $\forall v_x \in X(v_y, \mathcal{C}_i^{v_y})$, SET $HOPE\_MATRIX[x][y] \leftarrow 1$
      13. Update $DONE\_MATRIX$ /* as described in step (a) below */
          (a) $\forall v_z \in Y(v_y, \mathcal{C}_i^{v_y})$, SET $DONE\_MATRIX[y][z] \leftarrow 1$
      14. BREAK.
    end if
  end for
end for
for $y = 1$ to n **do**
  15. Compute $W(v_y, \mathcal{C}_i)$ /*as described in steps (a) and (b) below */
      (a) Initialize $W(v_y, \mathcal{C}_i) \leftarrow \emptyset$
      (b) $\forall v_x \in BNN_i[v_y]$, if $DONE\_MATRIX[x][y] = 1$, then
      SET $W(v_y, \mathcal{C}_i) \leftarrow W(v_y, \mathcal{C}_i) \cup \{v_x\}$
  16. SET $Y(v_y, \mathcal{C}_i) \leftarrow Y(v_y, \mathcal{C}_i^{v_1})$
end for
17. Return $\mathcal{C}_i$.

A.6 The proof of Lemma 10

Proof. Let $C_B(H)$ be the graph with $V(C_B(H)) = V(H)$ and $E(C_B(H)) = E(H) \cup \{(u,v) \mid u,v \in V_B\}$. In other words, $C_B(H)$ is obtained from H by making V_B a clique. Let H' be the graph with $V(H') = V(H)$ and $E(H') = E(H) \cup \{(u,v) \mid u \in V_A\}$. Observe that

$$H = C_B(H) \cap H'.$$

In conjunction with Lemma 1, this implies that

$$box(H) \leq box(C_B(H)) + box(H'). \tag{7}$$

▶ Claim 14. $box(C_B(H)) \leq 2box(S_B(H))$.

Proof of this claim is very similar to the proof of Lemma 3 in [6] and hence we only give a brief outline of it here. Assume $box(S_B(H)) = r$. Then by Lemma 1, there exist r interval graphs $I_1, \ldots, I_r$ such that $S_B(H) = I_1 \cap I_2 \cap \cdots \cap I_r$. For each $i \in [r]$, let f_i denote an interval representation of I_i. From these r interval graphs we construct $2r$ interval graphs $I'_1, I'_2, \ldots, I'_r, I''_1, I''_2, \ldots, I''_r$ as outlined below. Let f'_i, f''_i denote interval representations of I'_i and I''_i respectively, where $i \in [r]$.

Construction of f'_i :

$$\begin{aligned} \forall u \in V_A,\ f'_i(u) &= f_i(u). \\ \forall u \in V_B,\ f'_i(u) &= [\min_{v \in V_B}(L(f_i(v))), R(f_i(u))]. \end{aligned}$$

Construction of f''_i :

$$\begin{aligned} \forall u \in A,\ f''_i(u) &= f_i(u). \\ \forall u \in B,\ f''_i(u) &= [L(f_i(u)), \max_{v \in V_B}(R(f_i(v)))]. \end{aligned}$$

We leave it to the reader to verify that $C_B(H) = \bigcap_{i=1}^{r}(I'_i \cap I''_i)$.

▶ Claim 15. $box(H') \leq box(H_B)$.

Clearly, H' is obtained from H_B by adding universal vertices one after the other. Since adding a universal vertex to a graph does not increase its boxicity, $box(H') \leq box(H_B)$.

Combining Inequality 7, Claim 14 and Claim 15, we get $box(H) \leq 2box(S_B(H)) + box(H_B)$. ◀

Conditional Reactive Systems*

H. J. Sander Bruggink[1], Raphaël Cauderlier[2], Mathias Hülsbusch[1], and Barbara König[1]

1 **Universität Duisburg-Essen, Germany**
{sander.bruggink,mathias.huelsbusch,barbara_koenig}@uni-due.de
2 **ENS de Cachan, France**
rcauderl@dptinfo.ens-cachan.fr

Abstract

We lift the notion of nested application conditions from graph transformation systems to the general categorical setting of reactive systems as defined by Leifer and Milner. This serves two purposes: first, we enrich the formalism of reactive systems by adding application conditions for rules; second, it turns out that some constructions for graph transformation systems (such as computing weakest preconditions and strongest postconditions and showing local confluence by means of critical pair analysis) can be done very elegantly in the more general setting.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases reactive systems, graph transformation, graph logic, Hoare triples, critical pair analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.191

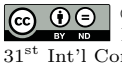
1 Introduction

Reactive Systems were introduced in [12, 11] to obtain a framework for deriving bisimulation congruences. They provide a general categorical setting for modelling abstract rewriting: both graph transformation systems and process calculi can be seen as special cases of reactive systems.

One of the main possibilities to control reductions is to fix a set of reactive contexts in which reductions are possible. However, this gives little control for complex specifications for which it is necessary to describe very precisely under which conditions a rule is applicable. Graph transformation systems provide the notion of negative application conditions or, more generally, nested application conditions [18, 6, 2, 7]. Nested application conditions are used extensively for specifications, for instance in (UML) model transformations.

Here we lift the idea of (application) conditions from the setting of graph transformation to reactive systems. This serves two purposes: first, we enrich the formalism of reactive systems by adding application conditions for rules. Second, it turns out that some constructions which are fairly cumbersome in the case of graph transformation (such as computation of weakest preconditions, strongest postconditions and critical pairs) become very elegant in this more abstract high-level setting. Interestingly it turns out that idem pushout squares (IPOs) [12], needed for label derivation in reactive systems, have a natural interpretation on application conditions and form the basis of an important construction, called shift or partial evaluation.

* Supported by the DFG project Behaviour-GT.



31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 191–203

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The paper is structured as follows: in Section 2 we introduce conditional reactive systems and boolean operations on conditions. Furthermore we briefly introduce adhesive categories and cospan categories. We then show in Section 3 that if conditions are interpreted in the base category, they are equivalent in expressiveness to a first-order logic on subobjects [1]. Section 4 introduces representative squares, a generalization of IPOs, and uses them in Section 5 in order to define logical operations such as shift and quantification on conditions. We study the properties of these operations, especially a characterization via adjunctions. Then in Section 6 we investigate applications, such as Hoare triples, weakest preconditions, strongest postconditions and a critical pair lemma for proving confluence.

We assume some basic knowledge of category theory.

2 Reactive Systems and Conditions

We now define the notion of reactive systems, first introduced in [12, 11] for label derivation and the definition of bisimulation congruences.

▶ **Definition 1** (Reactive System). Let $\mathbf{C}$ be a category with a distinguished object 0 (not necessarily initial). A *reactive system rule* is a pair $R = (\ell, r)$ of arrows – called left-hand side and right-hand side respectively – with $\ell, r \colon 0 \to I$ for some object I. Let $\mathcal{R}$ be a set of rules. We say that an arrow $a \colon 0 \to J$ *reduces* to $b \colon 0 \to J$ with the rules in $\mathcal{R}$ (in symbols: $a \Rightarrow_{\mathcal{R}} b$ or simply $a \Rightarrow b$) if there exists a rule $(r, \ell) \in \mathcal{R}$ with $\ell, r \colon 0 \to I$ and an arrow $c \colon I \to J$ such that $a = \ell\,;c$ and $b = r\,;c$.[1]

An important class of reactive systems can be defined over a base category $\mathbf{D}$ which has all pushouts along monos and in which pushouts preserve monos. Then we define as $\mathbf{C} = \mathit{ILC}(\mathbf{D})$ the category which has as objects the objects of $\mathbf{D}$ and as arrows cospans of the form $A \rightarrowtail^{f} B \leftarrow^{g} C$ (called *input-linear cospans*, because the left arrow f is a mono), where the middle object is taken up to isomorphism. Composition of cospans is performed via pushouts.

As base category $\mathbf{D}$ we will mainly use adhesive categories [10] where pushouts are well-behaved with respect to pullbacks and where, among other properties, monos are preserved by pushouts. Their properties make them suitable for deriving bisimulation congruences [20]. Here we need adhesive categories only in a limited way and the results can be stated and understood without the exact definition, which we therefore do not give. In some of the examples we will use the adhesive category $\mathbf{Graph_{fin}}$ which has finite graphs (with node and edge labels) as objects and graph morphisms as arrows. Reactive systems over $\mathit{ILC}(\mathbf{Graph_{fin}})$ coincide exactly with DPO graph transformation systems with injective matches (see [20]).

We will now define conditions, similar to the presentation in [18, 6], as tree-like structures, where nodes are annotated with quantifiers and objects and edges are annotated with arrows.

▶ **Definition 2** (Conditions). Let $\mathbf{C}$ be a category. A *condition* (in $\mathbf{C}$) is a triple $\mathcal{A} = (A, \mathcal{Q}, S)$ where

- A is an object of $\mathbf{C}$ (called the root object of $\mathcal{A}$ or $\mathrm{RO}(\mathcal{A})$),
- $\mathcal{Q}$ is a quantifier (either $\forall$ or $\exists$) and
- S is a set of pairs $(\mathcal{A}', f)$ such that $\mathcal{A}'$ is a condition and $f \colon A \to \mathrm{RO}(\mathcal{A}')$ is a $\mathbf{C}$-arrow.

[1] For arrows $f \colon A \to B$ and $g \colon B \to C$ we use the notation $f\,;g$ for their composition, that is $f\,;g \colon A \to C$.

The pair $(\mathcal{A}', f)$ will be denoted, by a slight abuse of notation, by $A \xrightarrow{f} \mathcal{A}'$. A condition $\mathcal{A}$ can be viewed as a tree, with $\mathrm{RO}(\mathcal{A})$ as the root and edges labelled with arrows. Note that in most cases we will require that S is a finite set, so that the trees are finitely branching.

▶ **Definition 3.** For all conditions $\mathcal{A}$, all objects C and all arrows $c\colon \mathrm{RO}(\mathcal{A}) \to C$ we define a satisfaction relation as follows:

$c \models (A, \forall, S)$ iff for every $(A \xrightarrow{f} \mathcal{A}') \in S$ and every arrow $\alpha\colon \mathrm{RO}(\mathcal{A}') \to C$ such that $f\,;\alpha = c$ we have $\alpha \models \mathcal{A}'$

$c \models (A, \exists, S)$ iff for some $(A \xrightarrow{f} \mathcal{A}') \in S$ there is an arrow $\alpha\colon \mathrm{RO}(\mathcal{A}') \to C$ with $f\,;\alpha = c$ and $\alpha \models \mathcal{A}'$

Now, given a reactive system over $\mathbf{C}$, it is natural to associate conditions with the target object of left-hand and right-hand sides and to interpret them on the contexts.

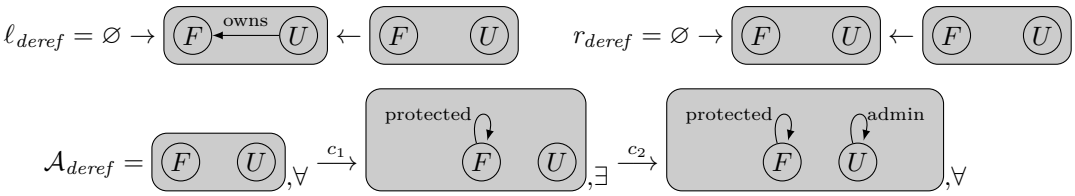
▶ **Definition 4** (Rules with Application Conditions)**.** Let $\mathbf{C}$ be a category with a distinguished object 0. A *rule with application condition* is a triple $(\ell, r, \mathcal{A})$ where $\ell, r\colon 0 \to I$ and $\mathcal{A}$ is a condition with root object I. We say that the rule is applicable to $a\colon 0 \to J$ whenever $a = \ell\,;c$ for some $c\colon I \to J$ such that $c \models \mathcal{A}$. The result of the rule application is $r\,;c$ as in Definition 1. Again we denote by $\Rightarrow_{\mathcal{R}}$ the rewriting relation induced by a set $\mathcal{R}$ of rules with application conditions.

▶ **Example 5.** For this example we are working in the category $ILC(\mathbf{Graph_{fin}})$.

Consider the following situation: In a file system a user can only dereference a file if it is owned by the user and the file is not marked as protected. To dereference protected files, the user must have administrator rights. Files which are no longer owned by any user are later removed by a garbage-collecting process.

Files are modelled by F-labeled nodes and users by U-labeled nodes. The fact that a user owns a files is represented by an edge which is labeled *owns* from the user to the file node. Protected files and users with administrator rights are designated with loops labeled *protected* and *admin*, respectively.

The derefencing is modelled by the rule $R_{deref} = (\ell_{deref}, r_{deref}, \mathcal{A}_{deref})$ with application condition $\mathcal{A}_{deref}$. The components are:



where $\varnothing$ is the empty graph and c_1 and c_2 are cospans where the right leg is the identity and the left morphism is induced by the labels. The condition expresses, that for all matches such that a *protected*-loop is connected to the file node, a *admin*-loop is connected to the user node. This condition is equivalent to the description above. Applying the rule has the effect that the *owns*-label is removed.

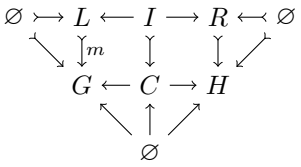
The condition for the garbage collection is:

$$\mathcal{B} = \varnothing, \exists \xrightarrow{c} \varnothing, \forall \quad \text{with } c = \varnothing \to \boxed{F} \leftarrow \varnothing,$$

which is true on exactly those cospans (with source object $\varnothing$) with isolated F-nodes in the center graph. In standard nested application conditions this requirement can be specified

only by saying that the node is not connected to any edge, which means quantifying over all edge labels. This does not yield a finite condition in the case of infinitely many edge labels [9].

Note that in graph transformation, nested application conditions are defined and handled in a slightly different way: they are defined in the base category instead of the cospan category and they are attached to the object L of the left-hand side, instead of the interface object I. In more detail: assume that $\mathbf{C} = ILC(\mathbf{D})$, hence rules have the form $\ell\colon \varnothing \rightarrowtail L \leftarrow I, r\colon \varnothing \rightarrowtail R \leftarrow I$ and we assume that contexts are of the form $c\colon I \rightarrowtail C \leftarrow \varnothing$. Note that the empty graph $\varnothing$ acts as the distinguished object 0. The conditions $a = \ell\,;c$, $b = r\,;c$ lead to a double-pushout diagram as shown on the right.



Now, a nested application condition in the sense of [6] is defined as a condition in $\mathbf{D}$ with root object L and it is evaluated on the match m. As above, a rule may only be applied if the condition is satisfied. This has some consequences. Most importantly, such nested application conditions are equal in expressivity to a first-order logic (see [18] for the case of graphs and Section 3 for the general case), whereas conditions on cospans are slightly more expressive: The paper [9] shows that every condition $\mathcal{A}$ in $\mathbf{D}$, consisting only of monos, can be translated into a condition $\mathcal{A}'$ in $\mathbf{C}$ such that $c\colon A \rightarrowtail B$ satisfies $\mathcal{A}$ iff $c\colon A \rightarrowtail B \leftarrow_{\mathrm{id}_B} B$ satisfies $\mathcal{A}'$. A translation in the other direction is in general impossible. We can always, even in the presence of infinitely many edge labels, specify that an isolated node exists; see also Example 5.

We feel that viewing conditions in the cospan category $\mathbf{C} = ILC(\mathbf{D})$ gives us an additional and very helpful layer of abstraction that simplifies many of the constructions to come. In Section 6 we will compare weakest preconditions and strongest postconditions as well as critical pair analysis in our setting with analogous notions in graph transformation and obtain constructions that are much simpler and straightforward. Hence we believe that switching to this higher level of abstraction is of great benefit.

We will in the following write $\mathcal{A} \models \mathcal{B}$ for two conditions $\mathcal{A}$, $\mathcal{B}$ with the same root object, whenever every arrow satisfying $\mathcal{A}$ satisfies $\mathcal{B}$ as well. We write $\mathcal{A} \equiv \mathcal{B}$ iff $\mathcal{A} \models \mathcal{B}$ and $\mathcal{B} \models \mathcal{A}$. We now define the usual boolean operations on conditions.

▶ **Definition 6** (Boolean Operations on Conditions)**.** We define operations on conditions as follows:

Constants: For an object A define $\mathit{false}_A := (A, \exists, \varnothing)$, $\mathit{true}_A := (A, \forall, \varnothing)$.

Negation: For a condition of the form $(A, \mathcal{Q}, S)$ with $\mathcal{Q} \in \{\forall, \exists\}$ we define:

$\neg(A, \forall, S) := (A, \exists, \{A \xrightarrow{f} \neg\mathcal{A}' \mid (A \xrightarrow{f} \mathcal{A}') \in S\})$

$\neg(A, \exists, S) := (A, \forall, \{A \xrightarrow{f} \neg\mathcal{A}' \mid (A \xrightarrow{f} \mathcal{A}') \in S\})$

Conjunction und Disjunction: For two conditions $\mathcal{A}, \mathcal{B}$ with root object C we define:

$\mathcal{A} \wedge \mathcal{B} := (C, \forall, \{C \xrightarrow{id_C} \mathcal{A}, C \xrightarrow{id_C} \mathcal{B}\})$

$\mathcal{A} \vee \mathcal{B} := (C, \exists, \{C \xrightarrow{id_C} \mathcal{A}, C \xrightarrow{id_C} \mathcal{B}\})$

These definitions are easily generalized to conjunctions and disjunctions over infinite sets of conditions.

▶ **Proposition 7** (Boolean Operations on Conditions)**.** *The operations and constants of Definition 6 satisfy the following laws:*

- *No arrow $c\colon A \to B$ satisfies $c \models \mathit{false}_A$ and every arrow $c\colon A \to B$ satisfies $c \models \mathit{true}_A$.*

- *For all conditions* $\mathcal{A}$ *and all arrows* $c\colon \mathrm{RO}(\mathcal{A}) \to B$*, we have* $c \models \neg\mathcal{A} \iff c \not\models \mathcal{A}$.
- *For all conditions* $\mathcal{A}, \mathcal{B}$ *with* $C = \mathrm{RO}(\mathcal{A}) = \mathrm{RO}(\mathcal{B})$ *and all arrows* $c\colon C \to D$ *it holds that* $(c \models \mathcal{A} \wedge \mathcal{B} \iff c \models \mathcal{A}$ *and* $c \models \mathcal{B})$ *and* $(c \models \mathcal{A} \vee \mathcal{B} \iff c \models \mathcal{A}$ *or* $c \models \mathcal{B})$.

3 Comparison to a First-Order Logic on Subobjects

Logic on subobjects [1] is a generalization of monadic second order logic of graphs to the world of categories. Nodes and edges are replaced by subobjects of fixed structure, sets of nodes and sets of edges are replaced by subobjects of arbitrary structure. Here we consider the first-order fragment of the logic on subobjects, which, as shown in [1], instantiates to first-order logic on graphs if we choose $\mathbf{Graph}_{\mathbf{fin}}$ as the underlying category.

In the following we will work in the subcategory $\mathbf{D}$ of an adhesive category $\mathbf{C}$, where $\mathbf{D}$ contains only monos. While this does not make a difference for the logic on subobjects, where we only quantify over monos anyway, for conditions it means that only monos are used in the evaluation, which is a typical restriction. (For an investigation of the effects of evaluating conditions on monos only see [5]. It is shown that under mild conditions both variants of conditions are expressively equivalent.) In addition the category $\mathbf{D}$ has a canonical embedding into $ILC(\mathbf{C})$. We remark again that, as discussed on Page 194, conditions in $ILC(\mathbf{C})$ are in general more expressive than conditions in the base category $\mathbf{D}$.

3.1 Syntax and semantics

We fix a category $\mathbf{C}$. The first-order logic of subobjects consists of *expressions* and *formulae*:

- Expressions are of the form $e = f \fatsemi x$, where x is a variable typed by an object A and f is a mono with codomain A. Expressions represent subobjects restricted by a mono.
- Formulae are generated by the grammar:

  $$\varphi ::= e \sqsubseteq e \mid \varphi \wedge \varphi \mid \neg\varphi \mid (\forall x\colon A)\ \varphi$$

  where x is a variable and A is an object. We use $e_1 = e_2$ as an abbreviation for $(e_1 \sqsubseteq e_2) \wedge (e_2 \sqsubseteq e_1)$. The notations $\varphi_1 \vee \varphi_2$, $\varphi_1 \to \varphi_2$ and $(\exists x\colon A)\ \varphi$ are defined in the usual way.

Let C be an object. A C-valuation maps variables to monos with codomain C (that is, to subobjects of C). We will overload the operator ; to the composition of C-valuations with monos, that is, for a C-valuation η and a mono $c\colon C \rightarrowtail D$, $\eta\,;c$ is a D-valuation defined as: $(\eta\,;c)(x) = \eta(x)\,;c$ for all x in the domain of η.

▶ **Definition 8.** For all objects C and all C-valuations η, we define a modelling relation as follows:

$C, \eta \models (f_1 \fatsemi x_1 \sqsubseteq f_2 \fatsemi x_2)$ iff $(f_1\,;\eta(x_1) \leq f_2\,;\eta(x_2))$.[2]
$C, \eta \models \varphi_1 \wedge \varphi_2$ iff $C, \eta \models \varphi_1$ and $C, \eta \models \varphi_2$.
$C, \eta \models \neg\varphi$ iff $C, \eta \not\models \varphi$.
$C, \eta \models (\forall x\colon T)\ \varphi$ iff for all monos $m\colon T \rightarrowtail C$ we have $C, \eta[x \mapsto m] \models \varphi$

[2] For monos $a\colon A \rightarrowtail T$ and $b\colon B \rightarrowtail T$ we write $a \leq b$ if there exists a mono $c\colon A \rightarrowtail B$ such that $a = c\,;b$.

3.2 From Conditions to the Logic on Subobjects

In this subsection we will translate conditions into the logic on subobjects.

▶ **Definition 9.** We define a translation from conditions to formulae as follows (where the variable x is of type A):

$$[\![(A, \forall, S)]\!](x) := \bigwedge_{(f,\mathcal{A}') \in S} (\forall x' \colon \mathrm{RO}(\mathcal{A}'))\ (f \fatsemi x' = x \ \to\ [\![\mathcal{A}']\!](x'))$$
$$[\![(A, \exists, S)]\!](x) := \bigvee_{(f,\mathcal{A}') \in S} (\exists x' \colon \mathrm{RO}(\mathcal{A}'))\ (f \fatsemi x' = x \ \wedge\ [\![\mathcal{A}']\!](x'))$$

A condition and its translation are equivalent in the following sense:

▶ **Proposition 10.** *If $\mathcal{A}$ is a condition then for all objects C and all monos $c \colon \mathrm{RO}(\mathcal{A}) \rightarrowtail C$ we have $c \models \mathcal{A}$ iff $C, (x \mapsto c) \models [\![\mathcal{A}]\!](x)$.*

3.3 From the Logic on Subobjects to Conditions

The idea behind the translation from formulae of the first-order logic of subobjects to conditions, is to consider all the different ways how the variables can overlap. That is, every time we quantify over a variable, we consider all possible ways how the objects pointed to by the new variable can overlap with the other variables, and build sub-conditions for all the possible overlaps. Formally, the set of overlaps of two objects A, B is defined as follows:

$$\mathrm{Ovl}(A, B) := \{(C, a, b) \mid a \colon A \rightarrowtail C \text{ and } b \colon B \rightarrowtail C \text{ are monos and jointly epi}\}$$

We assume in the following that $\mathrm{Ovl}(A, B)$ contains only one representative for each isomorphism class.

▶ **Definition 11.** Let B be an object and η a B-valuation. We define:

$$[\![f \fatsemi x \sqsubseteq g \fatsemi y]\!]^\eta_B := \begin{cases} \mathit{true}_B & \text{if } f\,;\eta(x) \leq g\,;\eta(y) \\ \mathit{false}_B & \text{if } f\,;\eta(x) \not\leq g\,;\eta(y) \end{cases}$$
$$[\![\varphi_1 \wedge \varphi_2]\!]^\eta_B := [\![\varphi_1]\!]^\eta_B \wedge [\![\varphi_2]\!]^\eta_B$$
$$[\![\neg\varphi]\!]^\eta_B := \neg[\![\varphi]\!]^\eta_B$$
$$[\![(\forall x \colon T)\ \varphi]\!]^\eta_B := \Big(B, \forall, \{B \overset{a}{\rightarrowtail} [\![\varphi]\!]^{\eta_{a,b}}_C \mid (C, a, b) \in \mathrm{Ovl}(T, B)\}\Big)$$
$$\text{where } \eta_{a,b}(y) = \begin{cases} a & \text{if } y = x \\ \eta(y)\,;b & \text{otherwise} \end{cases}$$

Note that the tree produced by Definition 11 is finitely branching (and because of the finite depth of formulas thus finite) if $\mathrm{Ovl}(A, B)$ is finite (up to isomorphism) for all objects A, B. In the category $\mathbf{Graph_{fin}}$ this is the case.

▶ **Proposition 12.** *Let $\mathbf{C}$ be an adhesive category. For any formula φ of the logic on subobjects, any objects B and C, any mono $c \colon B \rightarrowtail C$ and any B-valuation η it holds that $C, \eta\,; c \models \varphi$ if and only if $c \models [\![\varphi]\!]^\eta_B$.*

4 Representative Squares

We will now define the notion of representative squares, which describe representative ways to close a span of arrows. Such squares are intimately related to idem pushouts [12] or borrowed context diagrams [4].

▶ **Definition 13** (Representative classes of squares)**.** A class κ of commuting squares in a category $\mathbf{C}$ is called *representative* if κ satisfies the following property: for every commuting square of $\mathbf{C}$ (such as the one consisting of a, b, c', d' on the left) there exists a square in κ (consisting of a, b, c, d) and an arrow $e\colon M \to N$ which makes the diagram commute (on the right).

For two arrows $a\colon I \to J$, $b\colon I \to K$ we denote by $\kappa(a, b)$ the set of pairs (c, d) of arrows $c\colon J \to M$ and $d\colon K \to M$ such that a, b, c, d form a representative square in κ.

In the following, we fix a representative class κ of squares and we shall call every square in κ *representative*. Also note that the class of all squares of $\mathbf{C}$ is representative. The interesting classes of representative squares, however, have the property that $\kappa(a, b)$ is finite, which means that the constructions described below are effective since the finiteness of the transformed conditions is preserved.

Naturally, in a category with pushouts, pushouts are the most natural candidate for representative squares. Alternatively in adhesive categories they can be replaced by jointly epi squares. In [12] idem pushouts (IPOs) were introduced as a means to close squares in a representative way. (IPOs satisfy more properties than required in Definition 13, but those are not needed here.) A concrete manifestation of IPOs (or rather groupoidal idem pushouts or GIPOs) are borrowed context squares in the category $ILC(\mathbf{D})$, where the base category $\mathbf{D}$ is adhesive [20, 19].

For many categories of interest – such as the category of finite graphs (and graph morphisms) and the category of (input-linear) cospans of finite graphs – we can indeed guarantee a choice of κ such that each set $\kappa(a, b)$ is finite.

5 Shift and Quantification

One central operation is the shift of a condition along an arrow. The name shift is taken from an analogous operation for nested application conditions (see [13]). Intuitively a shift corresponds to a partial evaluation, where we assume that the arrows on which the condition is to be evaluated are of the form $\varphi\,;c$ for a fixed φ.

▶ **Definition 14** (Shift of a Condition)**.** Given a fixed set κ of representative squares, the shift $\mathcal{A}_{\downarrow\varphi}$ of a condition $\mathcal{A} = (A, \mathcal{Q}, S)$ along an arrow $\varphi\colon A \to B$ is inductively defined as follows:

$$\mathcal{A}_{\downarrow\varphi} = (B, \mathcal{Q}, \{(B \xrightarrow{\beta} \mathcal{A}'_{\downarrow\alpha}) \mid (A \xrightarrow{f} \mathcal{A}') \in S,\ (\alpha, \beta) \in \kappa(f, \varphi)\})$$

Note that the shift of a condition depends on the specific representive class of squares chosen. For different representative squares, the constructed condition can be different

(but equivalent). If we require that each set $\kappa(f, \varphi)$ is finite, then every finite condition is transformed into a finite-branching and hence again finite condition.

▶ **Proposition 15** (Shift). *Given two arrows $\varphi\colon A \to B$ and $c\colon B \to C$, and a condition $\mathcal{A}$ with root object A, the following holds:*

$$\varphi\,;c \models \mathcal{A} \iff c \models \mathcal{A}_{\downarrow\varphi}$$

We also define the following two quantification operations on conditions:

▶ **Definition 16** (Quantification). Given a condition $\mathcal{B}$ with root object B and an arrow $\varphi\colon A \to B$ we define for $\mathcal{Q} \in \{\exists, \forall\}$:

$$\mathcal{Q}\varphi.\mathcal{B} = (A, \mathcal{Q}, \{A \xrightarrow{\varphi} \mathcal{B}\})$$

We can view the set $\mathbf{C}_A$ of all conditions over a root object A as a category with an arrow between $\mathcal{A}$ and $\mathcal{B}$ whenever $\mathcal{A} \models \mathcal{B}$. Now the three operations on conditions can be seen as functors between these categories. Furthermore for $\varphi\colon A \to B$ it can be shown that $\exists\varphi\colon \mathbf{C}_B \to \mathbf{C}_A$ is the left adjoint of $\_{\downarrow\varphi}\colon \mathbf{C}_A \to \mathbf{C}_B$ and $\forall\varphi\colon \mathbf{C}_B \to \mathbf{C}_A$ is its right adjoint. These properties can be spelled out as follows.

▶ **Proposition 17** (Adjunction). *Let $\mathcal{A}, \mathcal{B}$ be two conditions with root object A, $\mathcal{C}, \mathcal{D}$ two conditions with root object B and let $\varphi\colon A \to B$. Then it holds that:*

1. *$\mathcal{A} \models \mathcal{B}$ implies $\mathcal{A}_{\downarrow\varphi} \models \mathcal{B}_{\downarrow\varphi}$.*
2. *$\mathcal{C} \models \mathcal{D}$ implies $\mathcal{Q}\varphi.\mathcal{C} \models \mathcal{Q}\varphi.\mathcal{D}$ for $\mathcal{Q} \in \{\exists, \forall\}$.*
3. *$\exists\varphi.(\mathcal{A}_{\downarrow\varphi}) \models \mathcal{A}$ and for every $\mathcal{C}$ with $\exists\varphi.\mathcal{C} \models \mathcal{A}$ we have that $\mathcal{C} \models \mathcal{A}_{\downarrow\varphi}$.*
4. *$\mathcal{A} \models \forall\varphi.(\mathcal{A}_{\downarrow\varphi})$ and for every $\mathcal{C}$ with $\mathcal{A} \models \forall\varphi.\mathcal{C}$ we have that $\mathcal{A}_{\downarrow\varphi} \models \mathcal{C}$.*

The adjunction is strongly reminiscent of categorical logic [14], where logical quantifiers are obtained as left or right adjoints to projections or pullback functor.

One easily obtains the following functoriality and de Morgan laws for shift and quantification:

$$\begin{array}{ccc}
\mathcal{A}_{\downarrow\mathrm{id}} \equiv \mathcal{A} & \mathcal{A}_{\downarrow\varphi\,;\psi} \equiv (\mathcal{A}_{\downarrow\varphi})_{\downarrow\psi} & \neg\mathcal{A}_{\downarrow\varphi} \equiv (\neg\mathcal{A})_{\downarrow\varphi} \\
\forall\mathrm{id}.\mathcal{A} \equiv \mathcal{A} & \forall(\varphi\,;\psi).\mathcal{A} \equiv \forall\varphi.\forall\psi.\mathcal{A} & \neg\forall\varphi.\mathcal{A} \equiv \exists\varphi.(\neg\mathcal{A}) \\
\exists\mathrm{id}.\mathcal{A} \equiv \mathcal{A} & \exists(\varphi\,;\psi).\mathcal{A} \equiv \exists\varphi.\exists\psi.\mathcal{A} & \neg\exists\varphi.\mathcal{A} \equiv \forall\varphi.(\neg\mathcal{A})
\end{array}$$

Since shift has a left *and* a right adjoint, it is a right *and* left adjoint itself. Left adjoints preserve colimits and right adjoints preserve limits. Since conjunction is a product, hence a limit, and disjunction is a coproduct, hence a colimit, we immediately obtain the following laws (which would not be very difficult to prove directly). This is in accordance with predicate logic where universal quantification distributes over conjunction and existential quantification over disjunction.

$$\begin{array}{cc}
(\mathcal{A} \wedge \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \wedge \mathcal{B}_{\downarrow\varphi} & (\mathcal{A} \vee \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \vee \mathcal{B}_{\downarrow\varphi} \\
\forall\varphi.(\mathcal{A} \wedge \mathcal{B}) \equiv \forall\varphi.\mathcal{A} \wedge \forall\varphi.\mathcal{B} & \exists\varphi.\mathcal{A} \vee \exists\varphi.\mathcal{B} \equiv \exists\varphi.\mathcal{A} \vee \exists\varphi.\mathcal{B}
\end{array}$$

Instead if we combine existential quantification and conjunction or universal quantification and disjunction we obtain the following laws involving representative squares.

▶ **Proposition 18** (Quantifier Distribution). *The following quantifier distribution laws hold:*

$$\exists\varphi.\mathcal{A} \wedge \exists\psi.\mathcal{B} \equiv \bigvee_{(\alpha,\beta)\in\kappa(\varphi,\psi)} \exists(\varphi;\alpha).(\mathcal{A}_{\downarrow\alpha} \wedge \mathcal{B}_{\downarrow\beta})$$

$$\forall\varphi.\mathcal{A} \vee \forall\psi.\mathcal{B} \equiv \bigwedge_{(\alpha,\beta)\in\kappa(\varphi,\psi)} \forall(\varphi;\alpha).(\mathcal{A}_{\downarrow\alpha} \vee \mathcal{B}_{\downarrow\beta})$$

6 Applications

After introducing the theory we will now give two applications: Hoare logic and critical pair analysis. Both applications generalize the special case of graph transformation to the setting of reactive systems.

Compared to earlier work on graph transformation our presentation is much simpler (compare with [13] for Hoare logic and [3] for critical pair analysis), which is due to the switch to a higher level of abstraction. Note that, as explained in Section 2, we are also working with a slightly more expressive logic.

Furthermore we improve the result in [15] by exhibiting an if-and-only-if result for critical pair analysis without application conditions and we strengthen the theorem in [3] for critical pair analysis with application conditions.

6.1 Weakest Preconditions and Strongest Postconditions

We will now show how to define Hoare triples, weakest preconditions and strongest postconditions in this framework.

▶ **Definition 19** (Hoare Triple, Weakest Precondition, Strongest Postcondition). Let $R = (\ell, r, \mathcal{C})$ be a rule with $\ell, r\colon 0 \to I$ and application condition $\mathcal{C}$ and let $\mathcal{A}, \mathcal{B}$ be conditions with root object 0. We say that $\mathcal{A}, R, \mathcal{B}$ form a *Hoare triple* (written as $\{\mathcal{A}\}\, R\, \{\mathcal{B}\}$) if for all $a, b\colon 0 \to J$ with $a \models \mathcal{A}$ and $a \Rightarrow_{\{R\}} b$ we have that $b \models \mathcal{B}$.

A condition $\mathcal{A}$ is called a *precondition* for R and $\mathcal{B}$ whenever $\{\mathcal{A}\}\, R\, \{\mathcal{B}\}$. Similarly $\mathcal{B}$ is called a *postcondition* for $\mathcal{A}$ and R.

A condition $\mathcal{A}$ is the *weakest precondition* for R and $\mathcal{B}$ (written $wp(R, \mathcal{B})$), whenever it is a precondition and for every other precondition $\mathcal{A}'$ we have that $\mathcal{A}' \models \mathcal{A}$. A condition $\mathcal{B}$ is the *strongest postcondition* for $\mathcal{A}$ and R (written $sp(\mathcal{A}, R)$), whenever it is a postcondition and for every other postcondition $\mathcal{B}'$ we have that $\mathcal{B} \models \mathcal{B}'$.

With all the machinery in place it is now easy to construct weakest preconditions and strongest postconditions.

▶ **Proposition 20** (Weakest Precondition, Strongest Postcondition). *Let $R = (\ell, r, \mathcal{C})$ be a rule with application condition as in Definition 19 and let $\mathcal{A}, \mathcal{B}$ be conditions with root object 0. Then*

- $wp(R, \mathcal{B}) = \forall\ell.(\mathcal{C} \to \mathcal{B}_{\downarrow r})$
- $sp(\mathcal{A}, R) = \exists r.(\mathcal{C} \wedge \mathcal{A}_{\downarrow\ell})$

Compared to the constructions for graph transformation or transformation systems over adhesive categories in [13] our definitions are much simpler. This is due mainly to two reasons: First, the shift operation relies on the powerful underlying notion of representative squares. Second, our conditions are defined in the same category as the rules, i.e., they would be cospans in the setting of [13]. As already discussed in [9] this simplifies matters and allows us to express dangling and inhibition conditions directly in the logics. When spelled out,

the constructions are more or less identical, but we believe that this more abstract view is very helpful to better understand the theory and to find additional applications, such as the following critical pair lemma.

6.2 Critical Pair Lemma

In order to show the fact that a given reactive system with rule set $\mathcal{R}$ is confluent, we use the well-known result from rewriting theory that states that a terminating rewriting system is confluent if and only if it is locally confluent [21]. Local confluence means that for all arrows a, b_1, b_2 with $a \Rightarrow b_1$, $a \Rightarrow b_2$ there exists an arrow c such that $b_1 \Rightarrow^* c$ and $b_2 \Rightarrow^* c$.

Local confluence can be reduced to showing confluence for so-called critical pairs, i.e., overlapping left-hand sides. Overlaps of left-hand sides can be described by our notion of representative squares. When there are only finitely many representative squares in $\kappa(a, b)$, showing local confluence becomes a much easier task.

▶ **Definition 21** (Critical Pair)**.** Let $R_i = (\ell_i, r_i)$, for $i \in \{1, 2\}$ with $\ell_i, r_i \colon 0 \to I_i$ be two rules. A *critical pair* for R_1, R_2 is a pair (c_1, c_2) of arrows $c_1 \colon I_1 \to K$ and $c_2 \colon I_2 \to K$ such that $(\ell_1, \ell_2, c_1, c_2)$ is a representative square.

▶ **Proposition 22** (Local Confluence for Rules without Application Conditions)**.** *Let $\mathcal{R}$ be a set of rules without application conditions. Then $\Rightarrow_{\mathcal{R}}$ is locally confluent if and only if for every critical pair (c_1, c_2) for rules (ℓ_i, r_i), $i \in \{1, 2\}$, in $\mathcal{R}$ there exists an arrow d with $r_1 ; c_1 \Rightarrow^* d$ and $r_2 ; c_2 \Rightarrow^* d$.*

The above result is not directly comparable to Plump's results in [15, 16]. Plump establishes the "if" direction of a criticial pair lemma, but his notion of confluence is different. Our notion of confluence is connected to what Plump calls "strongly joinable", which means that the common reducts must not only be isomorphic, but also have the same interface. Interestingly, our notion of confluence is decidable for terminating graph transformation systems – since the set of critical pairs is finite and constructible, and for terminating graph transformation systems (strong) joinability of the critical pairs is trivially decidable – whereas Plump's notion of confluence is not.

In order to extend Proposition 22 to rules with application conditions, we first have to collect conditions over a reduction sequence.

▶ **Definition 23** (Conditions for Reductions)**.** Let $\mathcal{R}$ be a set of rules with application conditions. For two arrows $a, b \colon 0 \to J$ we define $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ (where $\mathcal{A} \in \mathbf{C}_J$) if for all $d \colon J \to K$ with $d \models \mathcal{A}$ there exists $(\ell, r, \mathcal{B}) \in \mathcal{R}$ and an arrow c such that $a = \ell ; c$, $b = r ; c$ and $c ; d \models \mathcal{B}$.

The intuitive meaning of $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ is that a can reduce to b whenever it is put into a passive context satisfying $\mathcal{A}$. It is easy to show that $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ and $d \models \mathcal{A}$ imply $a ; d \Rightarrow_{\mathcal{R}} b ; d$.

▶ **Lemma 24.** *Let $\mathcal{R}$ be a set of rules with application conditions and let $a, b \colon 0 \to J$ two arrows. Then the weakest condition $\mathcal{A}$ with $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ can be obtained as follows:*

$$\mathcal{A} = \bigvee \{ \mathcal{B}_{\downarrow c} \mid (\ell, r, \mathcal{B}) \in \mathcal{R},\ \ell ; c = a,\ r ; c = b \}$$

▶ **Proposition 25.** *Let $\mathcal{R}$ be a set of rules with application conditions. The following laws hold:*

$$\frac{}{\mathcal{A}_{\downarrow c} \rhd \ell\,;c \Rightarrow^*_{\mathcal{R}} r\,;c} \text{ if } (\ell, r, \mathcal{A}) \in \mathcal{R} \qquad \frac{\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b \quad \mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} b}{\mathcal{A} \vee \mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} b}$$

$$\frac{\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b \quad \mathcal{B} \rhd b \Rightarrow^*_{\mathcal{R}} c}{\mathcal{A} \wedge \mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} c} \qquad \frac{\mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} b \quad \mathcal{A} \models \mathcal{B}}{\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b}$$

If, for two arrows a, b, there are only finitely many derivation paths leading from a to b, then the rules above are complete. This is for instance the case if $\Rightarrow_{\mathcal{R}}$ is finitely branching and the system is terminating. Otherwise there is no guarantee that the weakest condition $\mathcal{A}$ satisfying $\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b$ is even expressible as a finite (first-order) condition.

▶ **Proposition 26** (Local Confluence for Rules with Application Conditions). *Let $\mathcal{R}$ be a set of rules with application conditions. Then $\Rightarrow_{\mathcal{R}}$ is locally confluent if, for every critical pair (c_1, c_2) of rules $(\ell_i, r_i, \mathcal{A}_i)$ $(i \in \{1, 2\})$ in $\mathcal{R}$, there exist arrows $d_1, \ldots, d_m$ and conditions $\mathcal{C}^1_1, \mathcal{C}^1_2, \ldots, \mathcal{C}^m_1, \mathcal{C}^m_2$ such that $\mathcal{C}^i_1 \rhd r_1\,;c_1 \Rightarrow^*_{\mathcal{R}} d_i$ and $\mathcal{C}^i_2 \rhd r_2\,;c_2 \Rightarrow^*_{\mathcal{R}} d_i$ and*

$$(\mathcal{A}_1)_{\downarrow c_1} \wedge (\mathcal{A}_2)_{\downarrow c_2} \models \bigvee_{i=1}^{m} (\mathcal{C}^i_1 \wedge \mathcal{C}^i_2).$$

That is, we have to show that the condition describing that both left-hand sides match (shifted to the common context) implies one of the conditions specifiying that the reduction sequences can again be joined. The ideas underlying this result are taken from [3]. Note however that our result is stronger, since we weaken the precondition: the precondition in [3], transferred to reactive systems, would require that there is a single d such that $\mathcal{C}_1 \rhd r_1\,;c_1 \Rightarrow^*_{\mathcal{R}} d$ and $\mathcal{C}_2 \rhd r_1\,;c_2 \Rightarrow^*_{\mathcal{R}} d$ and $(\mathcal{A}_1)_{\downarrow c_1} \wedge (\mathcal{A}_2)_{\downarrow c_2} \models \mathcal{C}_1 \wedge \mathcal{C}_2$.

In Proposition 26 it is probably hard to obtain "if and only if", due to the non-monotonicity of rules with application conditions. Consider the following example:

▶ **Example 27.** We perform rewriting of labelled sets (basically Petri nets), where the start set contains a single element labelled A. A can either be replaced by B or C (that is, we have a critical pair). Now both B and C can be rewritten to D, but only if no E is present. Hence the system as such is not (locally) confluent. If however we add a rule removing E's the system would become confluent, since we could remove all E's first. However, this could take an arbitrary number of steps since there could be arbitrarily many E's around.

The problem is also, in a sense, that by writing $\mathcal{A} \rhd a \Rightarrow b$ we talk about a *passive* context (satisfying $\mathcal{A}$), on which the conditions are evaluated, but which does not truly interact with a.

7 Conclusion

We have shown how reactive systems can be extended with conditions, generalizing well-known constructions and results (axioms, pre- and postconditions, critical pair lemma) to the very general setting of reactive systems.

With the computation of weakest preconditions and strongest postconditions we now have the means to do Hoare logic reasoning (similar to [17]) for graph transformation and, even more interesting, to set up a framework for counterexample-guided abstraction refinement (CEGAR) in the sense of [8].

Another question of future research is to determine whether the axioms presented in Section 5 can be extended to a complete set of axioms. The thesis by Pennemann [13] contains some interesting developments going in this direction, including tool support. However, note

that this question will for sure also depend on the category: it is known that for finite graphs the satisfiability problem is semi-decidable, while the validity problem is not, whereas it is exactly the other way around for arbitrary (finite and infinite) graphs.

Finally we want to extend the derivation of labels and generation of bisimulation congruences to the case of conditional reactive systems.

References

1 H.J.S. Bruggink and B. König. A logic on subobjects and recognizability. In *Proc. of IFIP-TCS '10*, volume 323 of *IFIP AICT*, pages 197–212. Springer, 2010.

2 H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.

3 H. Ehrig, A. Habel, L. Lambers, F. Orejas, and U. Golas. Local confluence for rules with nested application conditions. In *Proc. of ICGT '10*, pages 330–345. Springer, 2010. LNCS 6372.

4 H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *MSCS*, 16(6):1133–1163, 2006.

5 A. Habel and K.-H. Pennemann. Satisfiability of high-level conditions. In *Proc. of ICGT '06*, pages 430–444. Springer, 2006. LNCS 4178.

6 A. Habel and K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Stuctures in Computer Science*, 19:245–296, 2009.

7 R. Heckel and A. Wagner. Ensuring consistency of conditional graph rewriting - a constructive approach. In *Proc. of the Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation*, volume 2 of *ENTCS*, 1995.

8 T.A. Henzinger, R. Jhala, R. Majumdar, and K.L. McMillan. Abstractions from proofs. In *Proc. of POPL '04*, pages 232–244. ACM, 2004.

9 M. Hülsbusch. Application conditions for reactive systems with applications to bisimulation theory. In *ICGT 2010 – Doctoral Symposium*, ECEASST 38, 2011.

10 S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *RAIRO – Theoretical Informatics and Applications*, 39(3), 2005.

11 J.J. Leifer. *Operational congruences for reactive systems*. PhD thesis, University of Cambridge Computer Laboratory, September 2001.

12 J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR 2000*, pages 243–258. Springer, 2000. LNCS 1877.

13 K.-H. Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Universität Oldenburg, May 2009.

14 A.M. Pitts. Categorical logic. In *Handbook of Logic in Computer Science V*. Oxford University Press, 2001.

15 D. Plump. Hypergraph rewriting: Critical pairs and undecidability of confluence. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 15, pages 201–214. John Wiley, 1993.

16 D. Plump. Confluence of graph transformation revisited. In A. Middeldorp, V. van Oostrom, F. van Raamsdonk, and R. de Vrijer, editors, *Festschrift Jan Willem Klop*. Springer, 2005. LNCS 3838.

17 C.M. Poskitt and D. Plump. A Hoare calculus for graph programs. In *Proc. of ICGT '10*, pages 139–154. Springer, 2010. LNCS 6372.

18 A. Rensink. Representing first-order logic using graphs. In *Proc. of ICGT '04*, pages 319–335. Springer, 2004. LNCS 3256.

19 V. Sassone and P. Sobociński. Reactive systems over cospans. In *Proc. of LICS '05*, pages 311–320. IEEE, 2005.

20 P. Sobociński. *Deriving process congruences from reaction rules.* PhD thesis, Department of Computer Science, University of Aarhus, 2004.
21 Terese. *Term Rewriting Systems.* CTTCS 55. Cambridge University Press, 2003.

Transforming Password Protocols to Compose

Céline Chevalier[1], Stéphanie Delaune[1], and Steve Kremer[1,2]

1 **LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France, France**
2 **INRIA Nancy – Grand Est, France**

Abstract

Formal, symbolic techniques are extremely useful for modelling and analysing security protocols. They improved our understanding of security protocols, allowed to discover flaws, and also provide support for protocol design. However, such analyses usually consider that the protocol is executed in isolation or assume a bounded number of protocol sessions. Hence, no security guarantee is provided when the protocol is executed in a more complex environment.

In this paper, we study whether password protocols can be safely composed, even when a same password is reused. More precisely, we present a transformation which maps a password protocol that is secure for a single protocol session (a decidable problem) to a protocol that is secure for an unbounded number of sessions. Our result provides an effective strategy to design secure password protocols: (i) design a protocol intended to be secure for one protocol session; (ii) apply our transformation and obtain a protocol which is secure for an unbounded number of sessions. Our technique also applies to compose different password protocols allowing us to obtain both inter-protocol and inter-session composition.

1998 ACM Subject Classification D.4.6 Security and Protection

Keywords and phrases Security, cryptographic protocols, composition

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.204

1 Introduction

Password-based cryptographic protocols are a prominent means to achieve authentication or to establish authenticated, shared session keys, *e.g.* EKE [10], SPEKE [23], or the KOY protocol [24]. The advantage of such schemes is that they do not rely on a key infrastructure but only on a shared password, which is often human chosen or at least human memorable. However, such passwords are generally *weak* and may be subject to dictionary (also called guessing) attacks. In an *online* dictionary attack an adversary tries to execute the protocol for each possible password. While such attacks are difficult to avoid they can be made impracticable by limiting the number of password trials or adding a time-out of few seconds after a wrong password. In an *offline* guessing attack an adversary interacts with one or more sessions in a first phase. In a second, offline phase the attacker uses the collected data to verify each potential password. In this paper we concentrate on the second type of attacks.

It has been widely acknowledged that security protocol design is extremely error prone and rigorous security proofs are a necessity. Formal, symbolic models, in the vein of Dolev and Yao's seminal work [21], provide effective and often automated methods to find errors or prove protocols correct. While most of these methods focus on secrecy and authentication, resistance against offline guessing attacks has been considered in some works [26, 9, 17]. We will in particular focus on an elegant definition of resistance against offline guessing attacks by Corin *et al.* [17] which was introduced in the framework of the applied pi calculus [1] and for which tool support exists [11, 9].

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 204–216

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Nowadays, state-of-the-art protocol analysis tools are able to analyse a variety of protocols. However, this analysis is generally carried out in isolation, *i.e.*, analysing one protocol at a time. This is motivated by the fact that in models like the applied pi calculus, security properties, even if shown in isolation, hold in the presence of an arbitrary (public) environment. This is similar to *universal composition* (UC) [14] in computational models. However, these arbitrary environments are public, in the sense that they don't have access to the secrets of the protocol under analysis. This is of course necessary as otherwise a completely arbitrary environment could simply output all secret cryptographic key material and trivially break the protocol's security. While not sharing key material may be a reasonable hypothesis in some cases it is certainly not the case when we compose the same sessions of a same protocol or in a situation where the same password is used in different protocols — it is indeed unreasonable to assume that all users have different passwords for each application.

Our contributions

In this paper we propose a simple protocol transformation which ensures that a same password can safely be shared between different protocols. More precisely, our results can be summarized as follows. We use a safe transformation which replaces a weak password w by $\mathsf{h}(t, w)$ where t is some *tag* and h a hash function. Then, we show how to use this tagging technique to compose different protocols. Consider n password protocols such that each protocol resists separately against guessing attacks on w. When we instantiate the tag t to a unique protocol identifier *pid*, one for each of the n protocols, we show that the parallel composition of these tagged protocols resists against guessing attacks on w, where w is the password shared by each of these protocols. Next we show how to dynamically establish a session identifier *sid*. Instantiating the tag t by this session identifier allows us to compose different sessions of a same protocol. Hence it is sufficient to prove resistance against guessing attacks on a single session of a protocol to conclude that the transformed protocol resists against guessing attacks for an unbounded number of sessions. These techniques can also be combined into a tag which consists of both the protocol and session identifier obtaining both inter-protocol and inter-session composition. One may note that resistance against guessing attack is generally not the main goal of a protocol, which may be authentication or key exchange. It follows however from our proofs that trace properties such as authentication will also be preserved. Detailed proofs of our results can be found in [16].

Related Work

In recent years, compositional reasoning has received a lot of attention. Datta *et al.* [19] provide a general strategy whereas our composition result identifies a specific class of protocols that can be composed. In [22, 5, 18], some criteria are given to ensure that parallel and in some works sequential composition is safe. In [6] the issue of composition of sessions of a same protocol is addressed using a transformation similar to the one considered in this paper. None of these works considers password protocols and resistance to guessing attacks. Composition of different password protocols (but not of sessions of the same protocol) using a protocol identifier tag was shown in [20]. In this paper we generalize these results to allow composition of sessions of a same protocol. Moreover, the composition theorem given in [20] only applies to two protocols (and cannot be iterated). This shortcoming was overseen by the authors of [20] and we adapt their result to apply to an arbitrary number of protocols in parallel.

In computational models, Boyko *et al.* [13] presented a security model for password-based

key-exchange based on simulation proofs, ensuring security in case of composition. A more generic solution was proposed by Canetti *et al.* [15] who propose a protocol based on KOY, which is secure in the UC model [14]. This work has been extended to active adversaries [4], group key exchange [3] and to define distributed public-key cryptography from passwords in *e.g.* [2]. A main difference between works in the UC model and our work (besides the obvious differences between symbolic and computational models) is that in the UC model designers generally apply an "ad-hoc recipe" (often using "magical" session identifiers given by the framework) and show that one session of a protocol fulfills the given requirements. The UC theorem then ensures composition, *i.e.*, composition follows from the strong security definition which has to be proven. In our work we make explicit the construction of session identifiers in our transformation and prove that a generic protocol transformation can be used to achieve composition. Note, however, that despite this difference, both approaches share many essential ideas.

Finally, we may note that *tagging* is a well known technique. We have already mentioned its use to achieve some forms of composition [6, 18]. Other forms of tagging were used to ensure termination of a verification procedure [12], safely bound the length of messages [7] or obtain decidability for the verification of some classes of protocols [27].

2 Modeling Protocols

In this section, we recall the cryptographic process calculus defined in [20] for describing protocols. This calculus is a simplified version of the applied pi calculus [1]. In particular we only consider one channel, which is public (*i.e.* under the control of the attacker) and we only consider finite processes, *i.e.* processes without replication.

2.1 Messages

A protocol consists of some agents communicating on a network. The messages sent by the agents are modelled using an abstract term algebra. For this, we assume an infinite set of *names* $\mathcal{N}$, for representing keys, data values, nonces, and names of agents, and we assume a *signature* Σ, *i.e.* a finite set of *function symbols* such as senc and sdec, each with an arity. Given a signature Σ and an infinite set of variables $\mathcal{X}$, we denote by $\mathcal{T}(\Sigma)$ (resp. $\mathcal{T}(\Sigma, \mathcal{X})$) the set of *ground terms* (resp. *terms*) over $\Sigma \cup \mathcal{N}$ (resp. $\Sigma \cup \mathcal{N} \cup \mathcal{X}$). We write $fn(M)$ (resp. $fv(M)$) for the set of names (resp. variables) that occur in the term M. A *substitution* σ is a mapping from a finite subset of $\mathcal{X}$ called its domain and written $\mathrm{dom}(\sigma)$ to $\mathcal{T}(\Sigma, \mathcal{X})$. The application of a substitution σ to a term T is written $T\sigma$. We also allow *replacement of names by terms*: the term $M\{^N/_n\}$ is the term obtained from M after replacing any occurrence of the name n by the term N (assuming that n does not occur in N). We sometimes abbreviate the sequence of terms $t_1, \ldots, t_n$ by $\tilde{t}$ and write $\{^{\tilde{t}}/_{\tilde{x}}\}$ for $\{^{t_1}/_{x_1}, \ldots ^{t_n}/_{x_n}\}$.

To model algebraic properties of cryptographic primitives, we define an *equational theory* by a finite set E of equations $U = V$ with $U, V \in \mathcal{T}(\Sigma, \mathcal{X})$ such that U, V do not contain names. We define $=_{\mathsf{E}}$ to be the smallest equivalence relation on terms, that contains E and that is closed under application of function symbols and substitutions of terms for variables.

▶ **Example 1.** Consider the signature $\Sigma = \{\mathsf{sdec}, \mathsf{senc}, \langle\,\rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{exp}\}$. The function symbols sdec, senc, $\langle\,\rangle$ and exp of arity 2 represent respectively symmetric encryption and decryption, pairing as well as exponentiation. Functions proj_1 and proj_2 of arity 1 model projection of the first and the second component of a pair. As an example that will be useful

for modelling the SPEKE protocol [23], we consider the equational theory E, defined by the following equations:

$$\begin{array}{ll} \mathsf{sdec}(\mathsf{senc}(x,y),y) = x & \mathsf{proj}_i(\langle x_1,x_2\rangle) = x_i \qquad (i \in \{1,2\}) \\ \mathsf{senc}(\mathsf{sdec}(x,y),y) = x & \mathsf{exp}(\mathsf{exp}(x,y),z) = \mathsf{exp}(\mathsf{exp}(x,z),y) \end{array}$$

Let $T_1 = \mathsf{senc}(\mathsf{proj}_2(\langle a,b\rangle),k)$ and $T_2 = \mathsf{senc}(b,k)$. We have that the terms T_1 and T_2 are equal modulo E, written $T_1 =_\mathsf{E} T_2$, while obviously the syntactic equality $T_1 = T_2$ does not hold.

To represent the knowledge of an attacker (who may have observed a sequence of messages $M_1,\ldots,M_\ell$), we use the concept of *frame*. A frame $\phi = \nu\tilde{n}.\sigma$ consists of a finite set $\tilde{n} \subseteq \mathcal{N}$ of *restricted* names (those unknown to the attacker), and a substitution σ of the form $\{{}^{M_1}/_{z_1},\ldots,{}^{M_\ell}/_{z_\ell}\}$ where each M_i is a ground term. The variables z_i enable an attacker to refer to each M_i. The *domain* of the frame ϕ, written $\mathrm{dom}(\phi)$, is $\mathrm{dom}(\sigma) = \{z_1,\ldots,z_\ell\}$.

Given a frame ϕ that represents the information available to an attacker, and an equational theory E on Σ, we may ask whether a given ground term M may be *deduced* from ϕ. This relation is written $\phi \vdash_\mathsf{E} M$ and is formally defined below.

▶ **Definition 2** (deduction)**.** Let M be a ground term and $\phi = \nu\tilde{n}.\sigma$ be a frame. We have that M is *deducible from* ϕ, denoted $\nu\tilde{n}.\sigma \vdash_\mathsf{E} M$, if and only if there exists a term $N \in \mathcal{T}(\Sigma,\mathcal{X})$ such that $fn(N) \cap \tilde{n} = \emptyset$ and $N\sigma =_\mathsf{E} M$. N is called a *recipe* of the term M.

Intuitively, the set of deducible messages is obtained from the messages M_i in ϕ, the names that are not restricted in ϕ, and closed under equality modulo E and application of function symbols.

▶ **Example 3.** Consider the theory E given in Example 1. Let $\phi = \nu b,k.\{{}^{\mathsf{senc}(b,k)}/_{z_1},{}^{k}/_{z_2}\}$. We have that $\phi \vdash_\mathsf{E} k$, $\phi \vdash_\mathsf{E} b$ and $\phi \vdash_\mathsf{E} a$. Indeed z_2, $\mathsf{sdec}(z_1,z_2)$ and a are recipes of the terms k, b and a respectively.

Two frames are considered equivalent when the attacker cannot detect the difference between the two situations they represent, that is, his ability to distinguish whether two recipes M, N produce the same term does not depend on the frame. Formally,

▶ **Definition 4** (static equivalence)**.** We say that two frames $\phi_1 = \nu\tilde{n}.\sigma_1$ and $\phi_2 = \nu\tilde{n}.\sigma_2$ are *statically equivalent*, $\phi_1 \approx_\mathsf{E} \phi_2$, when $\mathrm{dom}(\phi_1) = \mathrm{dom}(\phi_2)$, and for all terms M, N such that $fn(M,N) \cap \tilde{n} = \emptyset$, we have that: $M\sigma_1 =_\mathsf{E} N\sigma_1$ if, and only if, $M\sigma_2 =_\mathsf{E} N\sigma_2$.

Static equivalence is useful to model the notion of security we consider in this paper, namely *resistance against guessing attacks*. To resist against a guessing attack, the protocol must be designed such that the attacker cannot decide on the basis of the data collected whether his current guess of the password is the actual password or not. Assume $\phi = \nu\tilde{w}.\phi'$ is the frame representing the information gained by the attacker by eavesdropping one or more sessions and let $\tilde{w}$ be the sequence of weak passwords. The frame ϕ is resistant to guessing attacks if the attacker cannot distinguish between a situation in which he guesses the correct passwords $\tilde{w}$ and a situation in which he guesses incorrect ones, say $\tilde{w}'$.

▶ **Definition 5** (frame resistant to guessing attacks)**.** The frame $\nu\tilde{w}.\phi'$ is *resistant to guessing attacks* against the sequence of names $\tilde{w}$ if $\nu\tilde{w}.\phi' \approx \nu\tilde{w}.\nu\tilde{w}'.\phi'\{{}^{\tilde{w}'}/_{\tilde{w}}\}$ where $\tilde{w}'$ is a sequence of fresh names.

This definition was proposed in [17, 9]. A slightly simpler formulation requiring $\phi' \approx \phi'\{^{\tilde{w}'}/_{\tilde{w}}\}$ (without the name restrictions) was shown equivalent in [20] and will be used in this paper.

▶ **Example 6.** Consider the following protocol where h is a unary function symbol modelling a hash function (no equation on h):

$$\mathrm{A} \to \mathrm{B} : \mathsf{senc}(n, w) \qquad \mathrm{B} \to \mathrm{A} : \mathsf{senc}(\mathsf{h}(n), w)$$

An interesting problem arises if the shared key w is a weak secret, *i.e.* vulnerable to brute-force off-line testing. Indeed, the frame representing the knowledge of the attacker at the end of a normal execution of this protocol is $\phi = \nu w.\phi' = \nu w.\nu n.\{^{\mathsf{senc}(n,w)}/_{z_1}, {}^{M}/_{z_2}\}$ where:

$$M = \mathsf{senc}(\mathsf{h}(\mathsf{sdec}(\mathsf{senc}(n, w), w)), w) =_{\mathsf{E}} \mathsf{senc}(\mathsf{h}(n), w).$$

The frame ϕ is not resistant to guessing attacks against the password w. Indeed, the test $\mathsf{h}(\mathsf{sdec}(z_1, w)) \stackrel{?}{=} \mathsf{sdec}(z_2, w)$ is a witness of the non-equivalence $\phi' \not\approx_{\mathsf{E}} \phi'\{^{w'}/_{w}\}$.

2.2 Protocol Language and Semantics

Syntax

The grammar for processes is given below. One has plain processes P, Q, R and extended processes A, B, C that allow the use of active substitutions and restrictions.

$P, Q, R :=$	plain processes		$A, B, C :=$	extended processes	
	0	null process		P	plain processes
	$P \mid Q$	parallel composition		$A \mid B$	parallel composition
	$\mathrm{in}(x).P$	message input		$\nu n.A$	restriction
	$\mathrm{out}(M).P$	message output		$\{^{M}/_{x}\}$	active substitution
	if $M = N$ then P else Q	conditional			

As usual, names and variables have scopes, which are delimited by restrictions and inputs. We write $fv(A)$, $bv(A)$, $fn(A)$, $bn(A)$ for the sets of free and bound variables (resp. names). Moreover, we consider processes such that $bn(A) \cap fn(A) = \emptyset$, $bv(A) \cap fv(A) = \emptyset$, and each name and variable is bound at most once in A. An extended process is *closed* if all free variables are in the domain of an active substitution. An *instance* of an extended process is a process obtained by a bijective renaming of its bound names and variables. We observe that given processes A and B, there always exist instances A' and B' of A, respectively B, such that the process $A' \mid B'$ will respect the disjointness conditions on names and variables.

▶ **Example 7.** We illustrate our syntax with the SPEKE protocol (see [23] for a complete description).

$$\begin{aligned}
&\mathrm{A} \to \mathrm{B} : && M_1 = \mathsf{exp}(w, ra)\\
&\mathrm{B} \to \mathrm{A} : && M_2 = \mathsf{exp}(w, rb)\\
&\mathrm{A} \to \mathrm{B} : && M_3 = \mathsf{senc}(ca, \mathsf{exp}(\mathsf{exp}(w, rb), ra))\\
&\mathrm{B} \to \mathrm{A} : && M_4 = \mathsf{senc}(\langle ca, cb\rangle, \mathsf{exp}(\mathsf{exp}(w, ra), rb))\\
&\mathrm{A} \to \mathrm{B} : && M_5 = \mathsf{senc}(cb, \mathsf{exp}(\mathsf{exp}(w, rb), ra))
\end{aligned}$$

The goal of this protocol is to mutually authenticate A and B with respect to each other, provided that they share an initial secret w. This is done by a simple Diffie-Hellman exchange from a shared secret w, creating a common key $\mathsf{exp}(\mathsf{exp}(w, ra), rb) =_{\mathsf{E}} \mathsf{exp}(\mathsf{exp}(w, rb), ra)$, followed by a challenge-response transaction. The data ra, ca (resp. rb, cb) are nonces that

are freshly generated by A (resp. B). In our calculus, we model one session of the protocol as $\nu w.(A \mid B)$:

$$\begin{aligned} A = \nu ra, ca.&\mathrm{out}(\mathsf{exp}(w, ra)).\mathrm{in}(x_1).\\ &\mathrm{out}(\mathsf{senc}(ca, ka)).\mathrm{in}(x_2).\\ &\mathrm{out}(\mathsf{senc}(\mathsf{proj}_2(\mathsf{sdec}(x_2, ka)), ka)) \end{aligned}$$

$$\begin{aligned} B = \nu rb, cb.&\mathrm{in}(y_1).\mathrm{out}(\mathsf{exp}(w, rb)).\\ &\mathrm{in}(y_2).\mathrm{out}(\mathsf{senc}(\langle \mathsf{sdec}(y_2, kb), cb\rangle, kb)).\\ &\mathrm{in}(y_3).\ \mathrm{if}\ \mathsf{sdec}(y_3, kb) = cb\ \mathrm{then}\ P\ \mathrm{else}\ 0. \end{aligned}$$

where $ka = \mathsf{exp}(x_1, ra)$, $kb = \mathsf{exp}(y_1, rb)$, and P models an application that is executed when B has been successfully authenticated.

An *evaluation context* is an extended process with a hole instead of an extended process. Given an extended process A we denote by $\phi(A)$ the frame obtained by replacing any embedded plain processes in it with 0.

Semantics

We here only give an informal account of the semantics and refer the reader to [20] for the complete definition. We consider a basic *structural equivalence*, denoted $\equiv$, which includes for instance $A \mid B \equiv B \mid A$, $A \mid 0 \equiv A$ and $\nu n_1, n_2.A \equiv \nu n_2, n_1.A$. In particular, using structural equivalence, every extended process A can be rewritten to consist of a substitution and a plain process with some restricted names, *i.e.*,

$$A \equiv \nu \tilde{n}.(\{^{M_1}/_{z_1}\} \mid \ldots \mid \{^{M_k}/_{z_k}\} \mid P).$$

Moreover, any frame can be rewritten as $\nu n.\sigma$ matching the notion of frame introduced in Section 2.1.

Labelled operational semantics is the smallest relation $A \xrightarrow{\ell} A'$ between extended processes which is closed under structural equivalence ($\equiv$), application of evaluation context, and a few usual rules for input, output and conditional where ℓ is a label of one of the following forms:

- a label $\mathrm{in}(M)$, where M is a ground term such that $\phi(A) \vdash_{\mathsf{E}} M$;
- a label $\mathrm{out}(M)$, where M is a ground term, which corresponds to an output of M and which adds an active substitution $\{^M/_z\}$ in A';
- a label τ corresponding to a silent action (the evaluation of a conditional).

We denote by $\rightarrow$ the relation $\left\{\xrightarrow{\ell} \mid \ell \in \{in(M), out(M), \tau\},\ M \in \mathcal{T}(\Sigma)\right\}$ and by $\rightarrow^*$ its reflexive and transitive closure. Note that these semantics take the viewpoint that the attacker controls the entire network. Any message is sent to the attacker (who may or not forward it to the intended recipient) and the processes do not have any means to communicate directly.

▶ **Example 8.** We illustrate our semantics with the SPEKE protocol presented in Example 7. The derivation below represents a normal execution of the protocol. For simplicity of this example we suppose that $fv(P) = \emptyset$.

$$\begin{aligned} & \nu w.(A \mid B) \\ \xrightarrow{\mathrm{out}(\mathsf{exp}(w,ra))} \quad & \nu w, ra, ca.(\mathrm{in}(x_1).\mathrm{out}(\mathsf{senc}(ca, ka)).\mathrm{in}(x_2).\ldots \mid \{^{M_1}/_{z_1}\} \mid B) \\ \xrightarrow{\mathrm{in}(\mathsf{exp}(w,ra))} \quad & \nu w, ra, ca, rb, cb.(\mathrm{in}(x_1).\mathrm{out}(\mathsf{senc}(ca, ka)).\mathrm{in}(x_2).\ldots \mid \{^{M_1}/_{z_1}\} \mid B') \\ \rightarrow^* \quad & \nu w, ra, ca, rb, cb.(\{^{M_1}/_{z_1}, {}^{M_2}/_{z_2}, {}^{M_3}/_{z_3}, {}^{M_4}/_{z_4}, {}^{M_5}/_{z_5}\} \mid P) \end{aligned}$$

where B' represents the remaining actions of B in which y_1 is replaced by $\mathsf{exp}(w, ra)$, and $M_1, \ldots, M_5$ are defined in Example 7. The first step is an output of M_1 performed by A.

The active substitution $\{^{M_1}/_{z_1}\}$ allows the environment (*i.e.* the attacker) to access the message M_1 via the handle z_1. The handle z_1 is important since the environment cannot itself describe the term that was output, except by referring to it using z_1. Since M_1 is accessible to the environment via z_1, the next input action can be triggered: we have that $\nu w, ra, ca.\{^{M_1}/_{z_1}\} \vdash_{\mathsf{E}} \mathsf{exp}(w, ra)$ using the the recipe z_1.

In the remaining, we will focus our attention on password-based protocols.

▶ **Definition 9** (ℓ-party password protocol specification)**.** An *ℓ-party password protocol specification* $\mathcal{P}$ is a process such that:

$$\mathcal{P} = \nu w.(\nu \tilde{m}_1.P_1 \mid \ldots \mid \nu \tilde{m}_\ell.P_\ell)$$

where each P_i is a closed plain processes. The processes $\nu \tilde{m}_i.P_i$ are called the roles of $\mathcal{P}$.

The process $\nu w.(A \mid B)$ described in Example 7 is a 2-party password protocol specification with roles A and B. The notion of security we will mainly concentrate on is resistance against guessing attacks.

▶ **Definition 10** (process resistant to guessing attacks)**.** Let A be an extended, closed process and $\tilde{w} \subseteq bn(A)$. We say that a process A is *resistant to guessing attacks* against $\tilde{w}$ if, for every process B such that $A \to^* B$, we have that the frame $\phi(B)$ is resistant to guessing attacks against $\tilde{w}$.

3 Composition Results for Password-based Protocols

In this section, we present several composition results that hold for an arbitrary equational theory E. The only requirement we have is that there exists a function symbol h, which is a free symbol in E, *i.e.* h does not occur in any equation in E. Intuitively, h models a hash function.

3.1 Disjoint State

First, we note that, as usual, composition preserves security properties as soon as protocols have disjoint states, *i.e.*, they do not share any restricted names. Intuitively, this is due to the fact that when other protocols do not share any secrets of the analyzed protocol, then the attacker can completely simulate all messages sent by these other protocols. This has been formally shown in [20].

▶ **Theorem 11.** *[20] Let $A_1, \ldots, A_k$ be k extended processes such that for all i, we have that A_i is resistant to guessing attack against w_i. We have that $A_1 \mid \cdots \mid A_k$ is resistant to guessing attack against $w_1, \ldots, w_k$.*

3.2 Joint State

As soon as two protocols share a restricted name, *e.g.* a password, composition does not necessarily preserve security properties (see [20] for an example). We will use a tagging technique to avoid confusion between messages that come from different protocols. More precisely we will tag each occurrence of a password. Intuitively, we consider protocols that are well-tagged w.r.t. a secret w: all occurrences of w are of the form $\mathsf{h}(t, w)$ for some tag t.

Composing protocols

When each process is well-tagged with a different tag, it can be shown that the processes can be safely composed. One may think of these tags as protocol identifiers, which uniquely identify which protocol is executed, and avoid messages from different protocols to interfere with each other.

▶ **Theorem 12.** *Let $\alpha_1, \ldots, \alpha_k$ be k distinct names, and $\nu w.A_1, \ldots, \nu w.A_k$ be k processes such that $\alpha_i \notin bn(A_i)$ for any $i \in \{1, \ldots, k\}$. If each $\nu w.A_i$ is resistant to guessing attack against w then the process $\nu w.(A_1\{^{\mathsf{h}(\alpha_1,w)}/_w\} \mid \cdots \mid A_k\{^{\mathsf{h}(\alpha_k,w)}/_w\})$ is resistant to guessing attack against w.*

Actually, this result is a small adaptation from [20] (the result was shown for $k = 2$ only). This result can also be seen as a consequence of Proposition 15 and Lemma 16 (stated in Section 4) and a theorem showing that adding tags preserves resistance against guessing attacks (this last theorem is stated and proved in [20]).

The previous result is useful to compose distinct protocols. However, when we want to compose different sessions from the same protocol, we cannot assume that participants share a distinct tag for each possible session. In the following, we define a way to dynamically establish such a session tag.

Composing sessions from the same protocol

We now define a protocol transformation which establishes a dynamic tag that will guarantee composition. To establish such a tag that serves as a session identifier all participants generate a fresh nonce, that is sent to all other participants. This is similar to the establishment of session identifiers proposed by Barak [8]. The sequence of these nonces is then used to tag the password. Note that an active attacker may interfere with this initialization phase and may intercept and replace some of the nonces. However, since each participant generates a fresh nonce, these tags are indeed distinct for each session. This transformation is formally defined as follows.

▶ **Definition 13** (transformation $\overline{\mathcal{P}}$)**.** Let $\mathcal{P} = \nu w.(\nu \tilde{m}_1.P_1 \mid \ldots \mid \nu \tilde{m}_\ell.P_\ell)$ be a password protocol specification. Let $n_1, \ldots, n_\ell$ be fresh names and $\{x_i^j \mid 1 \leqslant i, j \leqslant \ell\}$ be a set of fresh variables. We define the protocol specification $\overline{\mathcal{P}} = \nu w.(\nu \tilde{m}_1, n_1.\overline{P_1} \mid \ldots \mid \nu \tilde{m}_\ell, n_\ell.\overline{P_\ell})$ as follows:

$$\overline{P_i} \;=\; \mathrm{in}(x_i^1).\ldots.\mathrm{in}(x_i^{i-1}).\mathrm{out}(n_i).\mathrm{in}(x_i^{i+1}).\mathrm{in}(x_i^\ell).P_i\{^{\mathsf{h}(tag_i,w)}/_w\}$$

where $tag_i = \langle x_i^1, \langle \ldots \langle x_i^{\ell-1}, x_i^\ell \rangle \rangle \rangle$ and $x_i^i = n_i$.

We can now state our composition result for sessions of a same protocol: if a protocol resists against guessing attacks on w then any number of instances of the transformed protocol will also resist to guessing attacks on w.

▶ **Theorem 14.** *Let $\mathcal{P} = \nu w.(\nu \tilde{m}_1, P_1 \mid \ldots \mid \nu \tilde{m}_\ell.P_\ell)$ be a password protocol specification that is resistant to guessing attacks against w. Let $\mathcal{P}'$ be such that $\overline{\mathcal{P}} = \nu w.\mathcal{P}'$, and $\mathcal{P}'_1, \ldots \mathcal{P}'_p$ be p instances of $\mathcal{P}'$. Then we have that $\nu w.(\mathcal{P}'_1 \mid \ldots \mid \mathcal{P}'_p)$ is resistant to guessing attacks against w.*

Discussion

Note that it is possible to combine these two ways of tagging. Applying successively the two previous theorems we obtain that a tag of the form $\mathsf{h}(\langle n_1, \ldots, n_\ell \rangle, \mathsf{h}(\alpha, w))$ allows to

safely compose different sessions of a same protocol, and also sessions of other protocols. It would also be easy to adapt the proofs to directly show that a simpler tag of the form $\mathsf{h}(\langle \alpha, \langle n_1, \ldots, n_\ell \rangle \rangle, w)$ could be used.

The notion of security we consider is resistance to guessing attacks. While generally resistance against guessing attacks is indeed a necessary condition to ensure security properties, this property is not a goal in itself. However, the way we prove our composition results allows us also to ensure that those protocols can be safely composed w.r.t. more classical trace-based security properties such as secrecy or authentication.

Finally, we note that our composition result yields a simple design methodology. It is sufficient to design a protocol which is secure for a single session. After applying the above protocol transformation we conclude that the transformed protocol is secure for an arbitrary number of sessions. Note that even though our protocol language does not include replication, our composition results for sessions ensure security for an unbounded number of sessions. Indeed, as any attack requires only a finite number of sessions, any attack on a transformed protocol which is secure for a single instance would yield a contradiction. As deciding resistance to guessing attacks is decidable for a bounded number of sessions (for a large class of equational theories) [9] our result can also be seen as a new decidability result for an unbounded number of sessions on a class of tagged protocols.

4 Proof of our main result

The goal of this section is to give an overview of the proof of Theorem 14. This proof is done in 4 main steps.

Step 1

Assume, by contradiction, that $P = \nu w.(\mathcal{P}'_1 \mid \ldots \mid \mathcal{P}'_p)$ admits a guessing attack on w. Hence there exists an attack derivation $P \rightarrow^* Q$ for some process Q such that $\phi(Q)$ is not resistant to a guessing attack against w.

Thanks to our transformation, we know that each role involved in P has to execute its preamble, *i.e.*, the preliminary nonce exchange of our transformation, at the end of which it computes a tag. Let $t_1, \ldots, t_k$ be the distinct tags that are computed during this derivation. Then, we group together roles (*i.e.* a process) that computed the same tag in order to retrieve a situation that is similar to when we use static tags. We note that the tags are constructed such that each group contains at most one instance of each role of $\overline{\mathcal{P}}$. Our aim is to show that an attack already exists on one of these groups, and so the attack is not due to composition. However, one difficulty comes from the fact that once the preambles have been executed, the tags that have been computed by the different roles may share some names in addition to w.

Step 2

The fact that some names are shared between the processes we would like to separate in order to retrieve the disjoint case significantly complicates the situation. Indeed, if composition still works, it is due to the fact that names shared among differently tagged processes only occur at particular positions. To get rid of shared names, we show that we can mimic a derivation by another derivation where tags $t_1, \ldots, t_k$ are replaced by constants $c_1, \ldots, c_k$ and different password are used ($w_1, \ldots, w_k$ instead of w). We denote by $\delta_{w_i,w}$ the replacement

$\{^{w}/_{w_1}\}\dots\{^{w}/_{w_k}\}$, by $\delta_{w_i,\mathsf{h}(c_i,w_i)}$ the replacement $\{^{\mathsf{h}(c_1,w_1)}/_{w_1}\}\dots\{^{\mathsf{h}(c_k,w_k)}/_{w_k}\}$ and by δ_{c_i,t_i} the replacement $\{^{t_1}/_{c_1}\}\dots\{^{t_k}/_{c_k}\}$.

▶ **Proposition 15.** *Let $t_1,\dots,t_k$ be distinct ground terms modulo E and $c_1,\dots,c_k,w_1,\dots,w_k$ be distinct fresh names. Let $\nu\tilde{n}.A$ be an extended process such that $bn(A)=\emptyset$, $w\notin fn(A)$, and $A=_{\mathsf{E}} A'\delta_{w_i,\mathsf{h}(c_i,w_i)}$ for some A' such that $c_1,\dots,c_k\notin fn(A')$. Moreover, we assume that $w,w_1,\dots,w_k,c_1,\dots,c_k\notin\tilde{n}$.*

Let $\overline{B}$ be such that $\nu w.\nu\tilde{n}.(A\delta_{c_i,t_i}\delta_{w_i,w})\xrightarrow{\ell}\overline{B}$. Moreover, when $\ell=in(\tilde{M})$ we assume that $w_1,\dots,w_k,c_1,\dots,c_k\notin fn(\tilde{M})$. Then there exists extended processes B, B', and labels ℓ_0, ℓ' such that:

- $\overline{B}\equiv\nu w.\nu\tilde{n}.(B\delta_{c_i,t_i}\delta_{w_i,w})$ *with* $bn(B)=\emptyset$ *and* $w\notin fn(B)$, $\ell=\ell_0\delta_{c_i,t_i}\delta_{w_i,w}$, *and*
- $B=_{\mathsf{E}} B'\delta_{w_i,\mathsf{h}(c_i,w_i)}$ *with* $c_1,\dots,c_k\notin fn(B')$, $\ell_0=_{\mathsf{E}}\ell'\delta_{w_i,\mathsf{h}(c_i,w_i)}$, *and*
- $\nu w_1\dots\nu w_k.\nu\tilde{n}.A\xrightarrow{\ell_0}\nu w_1\dots\nu w_k.\nu\tilde{n}.B$.

This proposition shows how to map an execution of $P\equiv\nu n_1\dots\nu n_k\nu w.(A_1\delta_{c_i,t_i}\delta_{w_i,w}\mid\dots\mid A_k\delta_{c_i,t_i}\delta_{w_i,w})$ (same password) to an execution of $\nu n_1\nu w_1.A_1\mid\dots\mid\nu n_k\nu w_k.A_k$ (different password) by maintaining a strong connection between these two derivations. Intuitively, the process $A_j\delta_{c_i,t_i}\delta_{w_i,w}$ contains the roles in P that computed the tag t_j in the attack derivation.

Note that, except for w, a name that is shared between $A_j\delta_{c_i,t_i}\delta_{w_i,w}$ and $A_{j'}\delta_{c_i,t_i}\delta_{w_i,w}$ ($j\neq j'$) necessarily occurs in a tag position in one of the process. Now that tags have been replaced by some constants, and the password w has been replaced by different passwords according to the tag, the processes A_j and $A_{j'}$ do not share any name.

This proposition is actually sufficient to establish that security properties, like authentication, are preserved by composition. However, to establish resistant against guessing attacks, we need more.

Step 3

We show that if a frame, obtained by executing several protocols that share a same password and that are tagged with terms t_i, is vulnerable to guessing attacks then the frame obtained by the corresponding execution of the protocols with different passwords and tagged with constants c_i is also vulnerable to guessing attacks.

▶ **Lemma 16.** *Let $t_1,\dots,t_k$ be distinct ground terms modulo E. Let $c_1,\dots,c_k,w_1,\dots,w_k$ be distinct fresh names, and $\phi=\nu\tilde{n}.\sigma$ be a frame such that $c_1,\dots,c_k,w_1,\dots,w_k\notin\tilde{n}$, and $\sigma=_{\mathsf{E}}\sigma_0\delta_{w_i,\mathsf{h}(c_i,w_i)}$ for some substitution σ_0. Let w be a fresh name, and $\psi=\nu\tilde{n}.(\sigma\delta_{c_i,t_i}\delta_{w_i,w})$. For each $1\leqslant i\leqslant k$, we also assume that $\nu w.\psi\vdash t_i$.*

If $\nu\tilde{w}.\phi$ is resistant to guessing attacks against $\tilde{w}=\{w_1,\dots,w_k\}$, then $\nu w.\psi$ is resistant to guessing attacks against w.

The proof of the lemma is technical because mapping all w_i's on the same password can introduce additional equalities between terms. However, each occurrence of the password is tagged, and the purpose of this design is to avoid the introduction of equalities between terms. Again, the lemma holds because the frames are well-tagged.

Thanks to Proposition 15 and Lemma 16 we obtain a guessing attack on the process $\nu n_1\nu w_1.A_1\mid\dots\mid\nu n_k\nu w_k.A_k$ against $w_1,\dots,w_k$.

Step 4

Applying Theorem 11 (combination for disjoint state protocols), we conclude that there is a guessing attack on $\nu n_i \nu w_i.A_i$ for some $i \in \{1, \ldots, k\}$. Then, it remains to show that the attack also works on the original protocol, *i.e.* the non-tagged version of the protocol. This is a direct application of Theorem 2 in [20]. This leads us to a contradiction since we have assumed that $\mathcal{P}$ is resistant to guessing attacks against w.

5 Conclusion

In this paper we propose a transformation for password protocols based on a simple tagging mechanism. This transformation ensures that security is preserved when protocols are composed with other protocols which may use the same password. We show that when protocols are tagged using a simple protocol identifier, we are able to compose different protocols. Computing a dynamic session identifier allows one to also compose different sessions of a same protocol. Hence, it is sufficient to prove that a protocol is secure for one session in order to conclude security under composition.

Currently, as stated, our composition results allow to preserve resistance against offline guessing attacks. As already discussed it also follows from our proofs that trace properties would be preserved. Formalizing for instance preservation of authentication should be a rather straightforward extension. A more ambitious direction for future work would be the composition of more general, indistinguishability properties, expressed in terms of observational equivalence. We also plan to investigate sufficient conditions to ensure composition of protocols in the vein of [25] avoiding to change existing protocols.

Acknowledgements

This work has been partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n^o 258865, project ProSecure and the ANR project JCJC VIP n^o 11 JS02 006 01.

References

1 M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

2 M. Abdalla, X. Boyen, C. Chevalier, and D. Pointcheval. Strong cryptography from weak secrets: Building efficient pke and ibe from distributed passwords in bilinear groups. In *Progress in Cryptology – AFRICACRYPT'10*. Springer, 2010.

3 M. Abdalla, C. Chevalier, L. Granboulan, and D. Pointcheval. UC-secure group key exchange with password-based authentication in the standard model. In *Proc. The Cryptographers' Track at the RSA Conference (CT-RSA'11)*, LNCS. Springer, 2011.

4 M. Abdalla, C. Chevalier, and D. Pointcheval. Smooth projective hashing for conditionally extractable commitments. In *Advances in Cryptology – CRYPTO'09*, volume 5677 of *LNCS*, pages 671–689. Springer, 2009.

5 S. Andova, C. Cremers, K. G. Steen, S. Mauw, S. M. lsnes, and S. Radomirović. Sufficient conditions for composing security protocols. *Information and Computation*, 206(2-4):425–459, 2008.

6 M. Arapinis, S. Delaune, and S. Kremer. From one session to many: Dynamic tags for security protocols. In *Proc. 15th International Conference on Logic for Programming,*

Artificial Intelligence, and Reasoning (LPAR'08), volume 5330 of *LNAI*, pages 128–142. Springer, 2008.

7 M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'07)*, volume 4855 of *LNCS*, pages 376–387. Springer, 2007.

8 B. Barak, Y. Lindell, and T. Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004.

9 M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM Press, 2005.

10 S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proc. Symposium on Security and Privacy (SP'92)*, pages 72–84. IEEE Comp. Soc., 1992.

11 B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *Proc. Symposium on Security and Privacy (SP'04)*, pages 86–100. IEEE Comp. Soc., 2004.

12 B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proc. Foundations of Software Science and Computation Structures (FoSSaCS'03)*, volume 2620 of *LNCS*, pages 136–152. Springer, 2003.

13 V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology – EUROCRYPT'00*, volume 1807 of *LNCS*, pages 156–171. Springer, 2000.

14 R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 136–145. IEEE Comp. Soc., 2001.

15 R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology – EUROCRYPT'05*, volume 3494 of *LNCS*, pages 404–421. Springer, 2005.

16 C. Chevalier, S. Delaune, and S. Kremer. Transforming password protocols to compose. Research Report LSV-11-21, Laboratoire Spécification et Vérification, ENS Cachan, France, Oct. 2011. 20 pages.

17 R. Corin, J. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. *ENTCS*, 121:47–63, 2005.

18 V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, Feb. 2009.

19 A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3), 2005.

20 S. Delaune, S. Kremer, and M. D. Ryan. Composition of password-based protocols. In *Proc. 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 239–251, Pittsburgh, PA, USA, June 2008. IEEE Computer Society Press.

21 D. Dolev and A. Yao. On the security of public key protocols. In *Proc. of the 22nd Symp. on Foundations of Computer Science*, pages 350–357. IEEE Comp. Soc. Press, 1981.

22 J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Comp. Soc. Press, 2000.

23 D. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, 1996.

24 J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Advances in Cryptology – EUROCRYPT'01*, volume 2045 of *LNCS*, pages 475–494. Springer, 2001.

25 R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. In *Proc. 18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM Press, 2011. To appear.

26 G. Lowe. Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.

27 R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

Obtaining a Bipartite Graph by Contracting Few Edges*

Pinar Heggernes[1], Pim van 't Hof[1], Daniel Lokshtanov[2], and Christophe Paul[3]

1 **Department of Informatics, University of Bergen, Norway.**
{pinar.heggernes|pim.vanthof}@ii.uib.no
2 **Department of Computer Science and Engineering, University of California San Diego, USA.**
dlokshtanov@cs.ucsd.edu
3 **CNRS, LIRMM, Université Montpellier 2, France.**
paul@lirmm.fr

Abstract

We initiate the study of the Bipartite Contraction problem from the perspective of parameterized complexity. In this problem we are given a graph G on n vertices and an integer k, and the task is to determine whether we can obtain a bipartite graph from G by a sequence of at most k edge contractions. Our main result is an $f(k)\,n^{O(1)}$ time algorithm for Bipartite Contraction. Despite a strong resemblance between Bipartite Contraction and the classical Odd Cycle Transversal (OCT) problem, the methods developed to tackle OCT do not seem to be directly applicable to Bipartite Contraction. To obtain our result, we combine several techniques and concepts that are central in parameterized complexity: iterative compression, irrelevant vertex, and important separators. To the best of our knowledge, this is the first time the irrelevant vertex technique and the concept of important separators are applied in unison. Furthermore, our algorithm may serve as a comprehensible example of the usage of the irrelevant vertex technique.

1998 ACM Subject Classification G.2.2, F.2.2

Keywords and phrases fixed parameter tractability, graph modification problems, edge contractions, bipartite graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.217

1 Introduction

Odd Cycle Transversal (OCT) is a central problem in parameterized complexity. The establishment of its fixed parameter tractability by Reed, Smith, and Vetta [24] in 2004, settling a long-standing open question [8], supplied the field with the powerful new technique of iterative compression [22]. Both OCT and the closely related Edge Bipartization problem take as input a graph G and an integer k, and ask whether a bipartite graph can be obtained by deleting at most k vertices, respectively k edges, from G. These two problems can be viewed as two ways of measuring how close G is to being bipartite. Over the last few years a considerable amount of research has been devoted to studying different measures of how close a graph is to being bipartite [9, 10, 15, 14], and how similarity to a bipartite graph can be exploited [6]. Another natural similarity measure is defined by the

* This work is supported by the Research Council of Norway.

Bipartite Contraction problem: Given a graph G and an integer k, can we obtain a bipartite graph from G by a sequence of at most k edge contractions in G? Considering the significant amount of interest the problems OCT and Edge Bipartization have received, we find it surprising that Bipartite Contraction has not yet been studied with respect to parameterized complexity. In this area, a graph problem with input G and k is said to be *fixed parameter tractable (FPT) when parameterized by* k if there is an algorithm with running time $f(k)\, n^{O(1)}$, where the function f depends only on k and not on the size of G.

The classical computational complexity of contracting at most k edges in a given graph to obtain a graph with a specific structure has been studied by Watanabe et al. [27, 28] and by Asano and Hirata [1]. NP-completeness of Bipartite Contraction follows from an easy polynomial-time reduction from Edge Bipartization, in which every edge of the input graph is replaced by a path of sufficiently large odd length. The study of edge contractions in general is motivated from Hamiltonian graph theory and graph minor theory, and it has applications in computer graphics and cluster analysis [18]. Graph minors play a central role in parameterized complexity, and the edge contraction operation in turn is essential in the study of graph minors: a graph H is a minor of a graph G if H can be obtained from G by a sequence of edge contractions, edge deletions, and vertex deletions. Although deciding whether a graph H is a minor of a given graph G is FPT when parameterized by the size of H [25], deciding whether H can be obtained from G by edge contractions is NP-complete already for some very small fixed graphs H, such as a path or a cycle on four vertices [2].

In this paper we show that Bipartite Contraction is FPT when parameterized by the number k of edges to be contracted. The key ingredients of our algorithm fundamentally differ from the ones used in the above-mentioned algorithms for OCT and Edge Bipartization. In the algorithm for OCT by Reed, Smith, and Vetta [24], iterative compression is combined with maximum flow arguments. The recent nearly linear time algorithm for the two problems, due to Kawarabayashi and Reed [14], uses the notion of odd minors, together with deep structural results of Robertson and Seymour [25] about graphs of large treewidth without large clique minors. Interestingly, Bipartite Contraction does not seem to be amenable to these approaches.

Although our algorithm is based on iterative compression, it seems difficult to adapt the compression step from [24] for OCT to work for Bipartite Contraction. Instead, we perform the compression step using a variant of the *irrelevant vertex technique*, introduced by Robertson and Seymour [25] (see also [26]). In particular, if the treewidth of the input graph is large, then we identify an irrelevant edge that can be deleted from the graph without affecting the outcome. The irrelevant vertex technique has played a key role in the solutions of several problems (see, e.g., [13, 15, 16]).

Our algorithm crucially deviates from previous work in the manner in which it finds the irrelevant edge. While previous work has relied on large minor models as obstructions to small treewidth, ours uses the fact that any graph of high treewidth contains a large *p-connected* set X [7]. A vertex set X is p-connected if, for any two subsets X_1 and X_2 of X with $|X_1| = |X_2| \leq p$, there are $|X_1|$ vertex-disjoint paths with one endpoint in X_1 and the other in X_2. Using p-connected sets in order to find irrelevant edges has several advantages. First, our algorithm avoids the huge parameter-dependence which seems to be an inadvertent side effect of applying Robertson and Seymour's graph minors machinery. Second, our arguments are nearly self-contained, and rely only on results whose proofs are simple enough to be taught in a graduate class.

Using p-connected sets in order to find an irrelevant vertex or edge is non-trivial, because p-connectivity is a more "implicit" notion than that of a large minor model. We overcome this

difficulty by using *important sets*. Important sets and the closely related notion of *important separators* were introduced in [20] to prove the fixed parameter tractability of multiway cut problems. The basic idea is that in many problems where terminals need to be separated in some way, it is sufficient to consider separators that are "as far as possible" from one of the terminals. Important separators turned out to be a crucial component, in some cases implicitly, in the solutions of cardinal problems in parameterized complexity [4, 5, 21, 23]. To the best of our knowledge, this is the first time the irrelevant vertex technique and important sets (or separators) are used together. We believe that this combination will turn out to be a useful and powerful tool.

2 Definitions and Notation

All graphs considered in this paper are finite, undirected, and simple, i.e., do not contain multiple edges or loops. Given a graph G, we denote its vertex set by $V(G)$ and its edge set by $E(G)$. We also use the ordered pair $(V(G), E(G))$ to represent G. We let $n = |V(G)|$ and $m = |E(G)|$. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *disjoint union* of G_1 and G_2 is the graph $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. The *deletion* of an edge $e \in E(G)$ yields the graph $G - e = (V(G), E(G) \setminus e)$. For a set $X \subseteq V(G)$, we write $G[X]$ to denote the subgraph of G *induced* by X. A graph is *connected* if there is a path between each pair of its vertices. The *connected components* of a graph are its maximal connected subgraphs. For any set $X \subseteq V(G)$, we write $\delta_G(X)$ to denote the set of edges in G that have exactly one endpoint in X. We define $d_G(X) = |\delta(X)|$.

The *contraction* of edge xy in G deletes vertices x and y from G, and replaces them by a new vertex, which is made adjacent to precisely those vertices that were adjacent to at least one of the vertices x and y. The resulting graph is denoted G/xy. Every edge contraction reduces the number of vertices in the graph by exactly one. We point out that several edges might disappear as the result of a single edge contraction. For a set $S \subseteq E(G)$, we write G/S to denote the graph obtained from G by repeatedly contracting an edge from S until no such edges remain. Let H be a graph with $V(H) = \{h_1, h_2, \ldots, h_\ell\}$. A graph G is *H-contractible* if H can be obtained from G by contracting edges. Saying that G is H-contractible is equivalent to saying that G has a so-called *H-witness structure* $\mathcal{W}$, which is a partition of $V(G)$ into *witness sets* $W(h_1), W(h_2), \ldots, W(h_\ell)$, satisfying the following properties: each witness set induces a connected subgraph of G, and for every two $h_i, h_j \in V(H)$, there is an edge in G between a vertex of $W(h_i)$ and a vertex of $W(h_j)$ if and only if h_i and h_j are adjacent in H. Let $G' = G[W(h_1)] \cup \cdots \cup G[W(h_\ell)]$ be the graph obtained from G by removing all the edges of G, apart from the ones that have both endpoints in the same witness set. In order to contract G to H, it is necessary and sufficient to contract all the edges of some spanning forest F of G'. Note that $|E(F)| = \sum_{i=1}^{\ell}(|W(h_i)| - 1) = |V(G)| - |V(H)|$.

A *2-coloring* of a graph G is a function $\phi : V(G) \to \{1, 2\}$. We point out that a 2-coloring of G is merely an assignment of colors 1 and 2 to the vertices of G, and should therefore not be confused with a *proper* 2-coloring of G, which is a 2-coloring with the additional property that no two adjacent vertices receive the same color. An edge uv is said to be *good* (with respect to ϕ) if $\phi(u) \neq \phi(v)$, and uv is called *bad* (with respect to ϕ) otherwise. A *good component* of ϕ is the vertex set of a connected component of the graph $(V(G), E')$, where $E' \subseteq E$ is the set of all edges that are good with respect to ϕ. Any 2-coloring ϕ of G defines a partition of $V(G)$ into two sets V_ϕ^1 and V_ϕ^2, which are the sets of vertices of G colored 1 and 2 by ϕ, respectively. A set $X \subseteq V(G)$ is a *monochromatic component* of ϕ if $G[X]$ is a connected component of $G[V_\phi^1]$ or a connected component of $G[V_\phi^2]$. We write

$\mathcal{M}_\phi$ to denote the set of all monochromatic components of ϕ. The *cost* of a 2-coloring ϕ is defined as $\sum_{X\in\mathcal{M}_\phi}(|X|-1)$. Note that the cost of a 2-coloring ϕ of G is 0 if and only if ϕ is a proper 2-coloring of G.

Let G be a graph. A *tree decomposition* of G is a pair $(T,\mathcal{X}=\{X_t\}_{t\in V(T)})$, where T is a tree and $\mathcal{X}$ is a collection of subsets of $V(G)$, satisfying the following three properties: (1) $\cup_{t\in V(T)}X_t = V$; (2) $\forall uv\in E(G), \exists t\in V(T) : \{u,v\}\subseteq X_t$; and (3) $\forall v\in V(G)$, $T[\{t : v\in X_t\}]$ is connected. The *width* of a tree decomposition is $\max_{t\in V(T)}|X_t|-1$ and the *treewidth* of G, denoted $tw(G)$, is the minimum width over all tree decompositions of G.

The syntax of *monadic second order (MSO) logic* of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall$, $\exists$ that can be applied to these variables, and the following five binary relations:

- $u\in U$, where u is a vertex variable and U is a vertex set variable;
- $d\in D$, where d is an edge variable and D is an edge set variable;
- $\mathbf{inc}(d,u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident to the vertex u;
- $\mathbf{adj}(u,v)$, where u and v are vertex variables and the interpretation is that u and v are adjacent;
- equality of variables representing vertices, edges, sets of vertices and sets of edges.

3 Bipartite Contraction and the Cost of 2-Colorings

In the Bipartite Contraction problem we are given a graph G and an integer k, and the task is to determine whether there exists a set $S\subseteq E(G)$ of at most k edges such that G/S is bipartite. The following lemma allows us to reformulate this problem in terms of 2-colorings of the graph G.

▶ **Lemma 1.** *A graph G has a 2-coloring ϕ of cost at most k if and only if there exists a set $S\subseteq E(G)$ of at most k edges such that G/S is bipartite.*

Proof. Suppose G has a 2-coloring ϕ of cost at most k. We build an edge set S as follows. For each monochromatic component $X\in\mathcal{M}_\phi$, find a spanning tree T_X of $G[X]$ and add the $|X|-1$ edges of T_X to S. The total number of edges in S is exactly the cost of ϕ, so $|S|\leq k$. It remains to argue that $G'=G/S$ is bipartite. Note that G' is obtained from G by contracting each spanning tree T_X to a single vertex t_X. Let ϕ' be the 2-coloring of G' that assigns to each $t_X\in V(G')$ the color of the corresponding monochromatic component $X\in\mathcal{M}_\phi$. Since each monochromatic component has been contracted to a single vertex, G' has no edge that is bad with respect to ϕ'. Hence ϕ' is a proper 2-coloring of G', implying that G' is bipartite.

For the reverse direction, suppose there is a set $S\subseteq E(G)$ of at most k edges such that $G'=G/S$ is bipartite. We define G^* to be the graph with the same vertex set as G and edge set S, i.e., $G^*=(V(G),S)$. Let $\mathcal{W}$ be the G'-witness structure of G whose witness sets are exactly the connected components of G^*. Let ϕ' be a proper 2-coloring of G'. We construct a 2-coloring ϕ of G as follows. For every $v\in V(G)$, we set $\phi(v)=\phi'(y)$, where y is the vertex in $V(G')$ such that $v\in W(y)$. Since the monochromatic components of ϕ are exactly the connected components of the graph G^*, and since G^* contains exactly $|S|$ edges, the cost of ϕ is at most $|S|\leq k$. ◀

An instance of the Cheap Coloring problem consists of a graph G and an integer k, and the task is to decide whether G has a 2-coloring of cost at most k. Lemma 1 shows that the problems Bipartite Contraction and Cheap Coloring are equivalent.

The deletion of an edge can not increase the cost of a 2-coloring, and can only decrease the cost of a 2-coloring by at most one. We state this as the following observation.

▶ Observation 1. Let ϕ be a 2-coloring of G of cost k. For any edge $uv \in E(G)$, the cost of ϕ in $G - uv$ is k or $k-1$.

Observation 1 allows us to use the well-known *iterative compression* technique of Reed, Smith and Vetta [24] to reduce the Cheap Coloring problem to the Cheaper Coloring problem. The Cheaper Coloring problem takes as input a graph G, an integer k, and a 2-coloring ϕ of G of cost $k+1$, and the task is to either find a 2-coloring of G of cost at most k, or to conclude that such a coloring does not exist.

▶ **Lemma 2.** *If there is an algorithm for* Cheaper Coloring *that runs in time* $f(k)\,n^c$ *for some constant* c*, then there is an algorithm for* Cheap Coloring *that runs in time* $f(k)\,n^c m$.

Proof. Suppose there exists an algorithm for Cheaper Coloring that runs in time $f(k)\,n^c$. Then we can solve an instance (G,k) of Cheap Coloring by iterating over the edges $e_1, e_2, \ldots e_m$ of G as follows. For every $i \in \{1,\ldots,m\}$, we define G_i to be the graph with vertex set $V(G)$ and edge set $E_i = \{e_j \ : \ j \leq i\}$. The graph G_1 has a 2-coloring ϕ_1 of cost 0, which is at most k. For the first k iterations, we trivially maintain a 2-coloring of cost at most k. Now, in iteration i of the algorithm, assume that we have a 2-coloring ϕ_i of cost at most k in G_i. By Observation 1, the cost of ϕ_i in G_{i+1} is at most $k+1$. If the cost of ϕ_i in G_{i+1} is at most k, then we proceed to iteration $i+1$. Otherwise, we run the algorithm for Cheaper Coloring with input (G_{i+1}, k, ϕ_i). If the algorithm concludes that G_{i+1} has no 2-coloring of cost at most k, then, by Observation 1, neither does G. If, on the other hand, the algorithm outputs a 2-coloring ϕ_{i+1} of G_{i+1} of cost at most k, then we proceed to the $(i+1)$th iteration. Since we call the algorithm for Cheaper Coloring at most m times, each time with parameter k, the time bound follows. ◀

We have now almost reached the variant of the problem that will be the focus of attention in the remainder of this paper. Given a graph G and two disjoint vertex sets $T_1, T_2 \subseteq V(G)$, a 2-coloring ϕ of G is a (T_1,T_2)-*extension* if ϕ colors every vertex in T_1 with 1 and every vertex in T_2 with 2. In the Cheap Coloring Extension problem we are given a *bipartite* graph G, two integers k and t, and two disjoint vertex sets T_1 and T_2 such that $|T_1|+|T_2| \leq t$; the sets T_1 and T_2 are not related to the sets of the bipartition of G. The objective is to find a (T_1,T_2)-extension ϕ of cost at most k, or to conclude that such a 2-coloring does not exist. We will say that a (T_1,T_2)-extension ϕ is a *cheapest* (T_1,T_2)-extension if there is no (T_1,T_2)-extension ϕ' with strictly lower cost than ϕ.

▶ **Lemma 3.** *If there is an algorithm for* Cheap Coloring Extension *that runs in time* $f(k,t)\,n^c$ *for some constant* c*, then there is an algorithm for* Cheaper Coloring *that runs in time* $4^{k+1} f(k, 2k+2)\,n^c$.

Proof. Given an $f(k,t)\,n^c$ time algorithm for Cheap Coloring Extension, we show how to solve an instance (G,k,ϕ) of Cheaper Coloring. Let S be the set of all bad edges in G with respect to ϕ, and let X be the set of endpoints of the edges in S. Since ϕ has cost $k+1$, we have $|X| \leq 2k+2$. We create 4^{k+1} instances of Cheap Coloring Extension as follows.

For every possible partition of X into two sets X_1 and X_2, we set $k' = k$ and $t = |X|$, and we build a graph $G(X_1,X_2)$ from G in the following way. As long as there is an edge $uv \in S$ such that u and v are both in X_1 or both in X_2, contract the edge uv, put the new vertex resulting from the contraction into the set X_i that u and v belonged to, and decrease

k' by 1. Since the cost of ϕ is at most $k+1$, we contract at most $k+1$ edges in this way, and hence $k' \geq -1$. When there are no such edges left, then we discard this partition of X into X_1 and X_2 if $k' = -1$; otherwise, we continue to build an instance of Cheap Coloring Extension as follows. Delete all edges $uv \in S$ with $u \in X_i$ and $v \in X_j$ such that $i \neq j$. Since S contains all the edges of G that are bad with respect to ϕ, and each of the edges of S is either contracted or deleted, the resulting graph $G(X_1, X_2)$ has no bad edges with respect to ϕ and is therefore bipartite. Thus we obtain an instance $(G(X_1, X_2), k', t, X_1, X_2)$ of Cheap Coloring Extension with $k' \geq 0$.

We now show that (G, k, ϕ) is a yes-instance of Cheaper Coloring if and only if there is a partition of X into X_1 and X_2 such that $(G(X_1, X_2), k', t, X_1, X_2)$ with $k' \geq 0$ is a yes-instance of Cheap Coloring Extension.

Suppose that (G, k, ϕ) is a yes-instance of Cheaper Coloring. Then there exists a 2-coloring ϕ^* of G of cost at most k. Let X_1 and X_2 be the vertices of X that are colored 1 and 2 by ϕ^*, respectively. Consider the set $S' \subseteq S$ of edges that were contracted in order to obtain $G(X_1, X_2)$ from G in the way described earlier. Since every edge in S' is bad with respect to ϕ^*, the cost of ϕ^* decreased by 1 with every edge contraction. Hence, ϕ^* is an (X_1, X_2)-extension of $G(X_1, X_2)$ of cost k'. We conclude that $(G(X_1, X_2), k', t, X_1, X_2)$ is a yes-instance of Cheap Coloring Extension.

For the reverse direction, suppose there is a partition of X into X_1 and X_2 such that $(G(X_1, X_2), k', t, X_1, X_2)$ is a yes-instance of Cheap Coloring Extension with $k' \geq 0$, i.e., the bipartite graph $G(X_1, X_2)$ has an (X_1, X_2)-extension ψ of cost at most k'. Let $S' \subseteq S$ be the set of edges that were contracted in G to create the instance $(G(X_1, X_2), k', t, X_1, X_2)$. Since $k' = k - |S'| \geq 0$, we have that $|S'| \leq k$. We define a 2-coloring θ of G by coloring both endpoints of every edge uv in S' with the color that ψ assigned to the vertex resulting from the contraction of the edge uv, and coloring all other vertices in G with the color they received from ψ. Clearly, the cost of θ is at most $k' + |S'| = k$, and therefore (G, k, ϕ) is a yes-instance of Cheaper Coloring.

Since we need to run the $f(k,t)\, n^c$ time algorithm for Cheap Coloring Extension at most 4^{k+1} times, with parameters $k' \leq k$ and $t = |X| \leq 2k + 2$ at each iteration, the time bound follows. ◀

The next section is devoted to showing that Cheap Coloring Extension is fixed parameter tractable when parameterized by k and t. The reason we want to work with the Cheap Coloring Extension problem rather than with the Bipartite Contraction problem directly is that, as we shall see in Section 4.2, Cheap Coloring Extension is a "cut" problem, and is therefore amenable to techniques based on *important separators* [20].

4 Solving Cheap Coloring Extension in FPT Time

In this section, we present an algorithm for the Cheap Coloring Extension problem. For the remainder of this section, let (G, k, t, T_1, T_2) be a given instance of Cheap Coloring Extension, where G is assumed to be connected. Recall that G is bipartite. The high level structure of our algorithm is as follows. If the treewidth of G is bounded by a function of k and t, then we can use standard dynamic programming techniques to solve the problem in time $f(k,t)\, n$. If, on the other hand, the treewidth of G is large, then we can find a large set of vertices which is "highly connected". In this case we show how to find in $f(k,t)\, n^{O(1)}$ time an edge $e \in E(G)$, such that G has a (T_1, T_2)-extension of cost at most k if and only if $G - e$ does. We then re-run our algorithm on $G - e$.

To make the distinction between the two cases in our algorithm more precise, we use the

following notion, due to Diestel et al. [7]. A set $X \subseteq V(G)$ is *p-connected* in G if $|X| \geq p$ and, for all subsets $X_1, X_2 \subseteq X$ with $|X_1| = |X_2| \leq p$, there are $|X_1|$ vertex-disjoint paths in G with one endpoint in X_1 and the other in X_2. Diestel et al. [7] prove the following statement in the proof of Proposition 3 (ii): if $h \geq p$ and G contains no p-connected set of size h, then G has treewidth $< h + p - 1$. (In fact, they prove a stronger version of this statement using the notion of an *externally* p-connected set, but we do not need this stronger assertion for our purposes.) We define a set X to be *well-connected* if it is $|X|/2$-connected. Using this definition, the result of Diestel et al. [7] can be seen to imply the following.

▶ **Theorem 4** ([7]). *If $tw(G) > w$, then G contains a well-connected set of size at least $2w/3$.*

The proof of Theorem 4 is constructive. In fact, given G and w, a tree decomposition of width at most w or a well-connected set of size at least $2w/3$ can be computed in time $c^w n^{O(1)}$ for some constant c [7]. We use Theorem 4 to compute either a tree-decomposition of G of width at most $3(4k^2)\,t\,4^{4k^2}+3$ or a well-connected set Y of size at least $2(4k^2)\,t\,4^{4k^2}+2$. Section 4.1 deals with the first case, whereas the second case is covered in Section 4.2.

4.1 Small Treewidth

Suppose our algorithm has found a tree-decomposition of G of width at most $3(4k^2)\,t\,4^{4k^2}+3$. We use the following celebrated theorem by Courcelle [3] to solve the Cheap Coloring Extension problem in this case.

▶ **Theorem 5** ([3]). *There is an algorithm that tests whether a monadic second order formula ψ holds on a graph G of treewidth w, in time $f(|\psi|, w)\,n$.*

Since Cheap Coloring Extension can be expressed in monadic second order logic (we omit the details due to page restrictions), we have the following result.

▶ **Lemma 6.** *There is an algorithm that, given an instance (G, k, t, T_1, T_2) of* Cheap Coloring Extension *together with a tree-decomposition of G of width w, solves the instance in time $f(k, t, w)\,n$.*

We would like to remark that, given an instance (G, k, t, T_1, T_2) of Cheap Coloring Extension together with a tree-decomposition of G of width w, it is possible to solve that instance in time $(w+1)^{O(w)}n$ using standard dynamic programming techniques, which gives a much faster algorithm than the one obtained by applying Theorem 5 on the monadic second order formula.

4.2 Large Treewidth and Irrelevant Edges

Suppose our algorithm did not find a tree-decomposition of G of small width, but instead found a well-connected set Y of size at least $2(4k^2)\,t\,4^{4k^2}+2$. We use Y throughout this section to refer to this specific set. An edge $e \in E(G)$ is said to be *irrelevant* if it satisfies the following property: G has a (T_1, T_2)-extension of cost at most k if and only if $G-e$ does. We will show that the presence of the large well-connected set Y guarantees the presence of an irrelevant edge e in G. Hence we find such an irrelevant edge e in G, delete it from the graph, and solve Cheap Coloring Extension on the instance $(G-e, k, t, T_1, T_2)$. Since each iteration of this process deletes an edge, we will find a tree-decomposition of the graph under consideration of small width after at most m iterations, in which case we solve the problem as described in Section 4.1.

▶ Observation 2. Let ϕ be a 2-coloring of G. No bad edge has both endpoints in the same good component of ϕ.

Proof. Suppose, for contradiction, that G has a bad edge uv such that both u and v belong to a good component C of ϕ. Since uv is bad, we have $\phi(u) = \phi(v)$. Every good component is connected, so there is a path P in C, starting in u and ending in v, consisting only of good edges. The path P must contain an even number of edges, implying that P and uv together form an odd cycle in G. This contradicts the assumption that G, which is part of the instance (G, k, t, T_1, T_2) of Cheap Coloring Extension that we are solving, is bipartite. ◀

▶ Observation 3. Let $uv \in E(G)$. If ϕ is a cheapest (T_1, T_2)-extension of $G - uv$ and u and v are in the same good component of ϕ, then uv is irrelevant.

Proof. Suppose u and v belong to the same good component of a cheapest (T_1, T_2)-extension ϕ of $G - uv$. Note that ϕ is a 2-coloring of G, and that the edge uv in G is good with respect to ϕ as a result of Observation 2. Hence ϕ is a (T_1, T_2)-extension of G, and the cost of ϕ in G equals the cost of ϕ in $G - uv$. As a result of Observation 1, ϕ must be a cheapest (T_1, T_2)-extension of G. Since the cost of a cheapest (T_1, T_2)-extension of $G - uv$ equals the cost of a cheapest (T_1, T_2)-extension of G, the edge uv is irrelevant by definition. ◀

In order to use Observation 3, we need to identify two adjacent vertices u and v in G that will end up in the same good component of some cheapest (T_1, T_2)-extension of $G - uv$. The vertices in Y are good candidates, because they are so highly connected to each other. Over the next few lemmas we formalize this intuition. We start with two observations that will allow us, in the proof of Lemma 7 below, to bound the number of bad edges and the number of good components of a cheapest (T_1, T_2)-extension of G of cost at most k.

▶ Observation 4. Let ϕ be a 2-coloring of G. If ϕ has cost at most k, then there are less than $2k^2$ bad edges.

Proof. Let $\mathcal{M}'_\phi = \{X \in \mathcal{M}_\phi \ : \ |X| \geq 2\}$ be the set of monochromatic components of ϕ containing more than one vertex, and let G' be the disjoint union of the graphs induced in G by the elements of $\mathcal{M}'_\phi$, i.e., $G' = \bigcup_{X \in \mathcal{M}'_\phi} G[X]$. By definition, the cost of ϕ is $\sum_{X \in \mathcal{M}_\phi}(|X|-1) = \sum_{X \in \mathcal{M}'_\phi}(|X|-1)$, which is exactly the number of edges in any spanning forest of G'. Since any forest on at most k edges without isolated vertices has at most $2k$ vertices, we have $|V(G')| \leq 2k$. Every bad edge has both endpoints in $V(G')$, so the number of bad edges is at most $\binom{2k}{2} = 2k^2 - k < 2k^2$. ◀

▶ Observation 5. Let ϕ be a cheapest (T_1, T_2)-extension of G. Every good component of ϕ contains a vertex from $T_1 \cup T_2$.

Proof. Suppose a good component C of ϕ does not contain any vertex from $T_1 \cup T_2$. We build a coloring ϕ' from ϕ by changing the color of every vertex in C, leaving the color of every other vertex unchanged, i.e., $\phi'(v) = 3 - \phi(v)$ if $v \in C$, and $\phi'(v) = \phi(v)$ if $v \notin C$. Since $\phi(v) = \phi'(v)$ for every $v \in T_1 \cup T_2$, ϕ' is a (T_1, T_2)-extension of G. Furthermore, every edge that was good with respect to ϕ is good with respect to ϕ', while every edge in $\delta_G(C)$ was bad with respect to ϕ and is good with respect to ϕ'. Recall that G is assumed to be connected. Hence there is some vertex $v \in C$ which is incident to at least one edge that was bad with respect to ϕ. On the other hand, all edges incident to v are good with respect to ϕ'. Hence $\{v\}$ is a monochromatic component of ϕ', but $\{v\}$ was not a monochromatic component of ϕ. This means that $|\mathcal{M}_\phi| < |\mathcal{M}_{\phi'}|$. This, together with the observation that the number of edges that are bad with respect to ϕ' is not higher than the number of edges

that were bad with respect to ϕ, implies that the cost of ϕ' is strictly less than the cost of ϕ. This contradicts the assumption that ϕ is a cheapest (T_1, T_2)-extension of G. ◀

The next lemma shows that almost all the vertices of Y appear in the same good component of any cheapest (T_1, T_2)-extension ϕ of G of cost at most k. In fact, we prove that the same holds if we remove any edge of G.

▶ **Lemma 7.** *Let $uv \in E(G)$ and let ϕ be a cheapest (T_1, T_2)-extension of $G - uv$. If ϕ has cost at most k, then there exists exactly one good component C^* of ϕ satisfying $|Y \setminus C^*| \leq 2k^2$, and every other good component C' of ϕ satisfies $|Y \cap C'| \leq 2k^2$.*

Proof. Suppose ϕ has cost at most k, and let C be a good component of ϕ. We first show that either $|Y \setminus C| \leq 2k^2$ or $|Y \cap C| \leq 2k^2$. Suppose for contradiction that $|Y \cap C| > 2k^2$ and $|Y \setminus C| > 2k^2$. We define Y_1 to be the smallest of the two sets $Y \cap C$ and $Y \setminus C$, and Y_2 to be any subset of the largest of the two sets such that $|Y_2| = |Y_1|$. Note that $2k^2 + 1 \leq |Y_1| = |Y_2| \leq |Y|/2$. By the definition of a well-connected set, there are $|Y_1| \geq 2k^2 + 1$ vertex-disjoint paths with one endpoint in $Y \cap C$ and the other in $Y \setminus C$. At least $2k^2$ of these paths exist in $G - uv$, and each of those must contain an edge in $\delta_{G-uv}(C)$. Since each edge in $\delta_{G-uv}(C)$ is bad, it follows that ϕ has at least $2k^2$ bad edges, contradicting Observation 4.

Now suppose for contradiction that ϕ does not have a good component C^* with $|Y \setminus C^*| \leq 2k^2$. Then $|Y \cap C| \leq 2k^2$ for every good component C of ϕ, as we showed earlier. Since ϕ has at most $t = |T_1| + |T_2|$ good components as a result of Observation 5, at most $t\,2k^2$ vertices of Y appear in good components. The fact that the size of Y is much larger than $t\,2k^2$, together with the observation that every vertex of G appears in a good component by definition, yields the desired contradiction. Hence we know that ϕ has a good component C^* with $|Y \setminus C^*| \leq 2k^2$. The uniqueness of C^* follows from the sizes of Y and C^*, and the fact that the good components of ϕ are pairwise disjoint. ◀

There are two problems with how to exploit the knowledge obtained from Lemma 7. The first is that, even though we know that almost all the vertices of Y appear in the same good component together, we do not know exactly which ones do. The second problem is that we are looking for an *edge* with both endpoints in the same good component, and Y could be an independent set and thus not immediately give us an edge to delete. We deal with both problems by employing the very useful notion of *important sets*. For two vertices $x, y \in V(G)$, we say that a set $X \subseteq V(G)$ is (x, y)-*important* if it satisfies the following three properties: (1) $x \in X$ and $y \notin X$; (2) $G[X]$ is connected; and (3) there is no $X' \supset X$, $y \notin X'$ such that $d_G(X') \leq d_G(X)$ and $G[X']$ is connected. The following theorem was first proved in [4]. We use here the formulation in [19], because that one best fits the purposes of this paper.

▶ **Theorem 8** ([4, 19])**.** *Let x, y be two vertices in a graph G. For every $p \geq 0$, there are at most 4^p (x, y)-important sets X such that $d_G(X) \leq p$. Furthermore, these important sets can be enumerated in time $4^p \cdot n^{O(1)}$.*

Suppose $G - uv$ has a cheapest (T_1, T_2)-extension ϕ of cost at most k for an edge $uv \in E(G)$. We will use the important sets together with Lemma 7 to identify vertices in Y which must be in the unique good component C^* of ϕ that contains all but at most $2k^2$ vertices of Y. We first build a graph G^* from G by adding a new vertex y^* and making y^* adjacent to all vertices in Y. We then enumerate all $x \in T_1 \cup T_2$ and all (x, y^*)-important sets X in G^* such that $d_{G^*}(X) \leq 4k^2$. By Theorem 8, this can be done in time $4^{4k^2} n^{O(1)}$ for each choice

of x. Finally, we define the set Z to be the union of all enumerated sets X. In other words,

$$Z = \{w \in V(G^*) : \exists x \in T_1 \cup T_2, X \subseteq V(G^*), w \in X, d_{G^*}(X) \leq 4k^2, X \text{ is } (x, y^*)\text{-important}\}$$

Observe that, given G and Y, Z can be computed in time $t\,4^{4k^2} n^{O(1)}$. We will use the set Z in the following way. First we show that if there is an edge $uv \in E(G)$ such that neither u nor v belongs to Z, then the edge uv is irrelevant. Then we show that such an edge always exists.

▶ **Lemma 9.** *Let $uv \in E(G)$ such that $u \notin Z$ and $v \notin Z$. Then uv is irrelevant.*

Proof. Let ϕ be a cheapest (T_1, T_2)-extension of $G - uv$, and suppose ϕ has cost at most k. Let C^* be a good component of ϕ such that $|Y \setminus C^*| \leq 2k^2$. By Lemma 7, such a component C^* exists, and every other good component C of ϕ satisfies $|Y \cap C| \leq 2k^2$. We prove that both u and v are in C^*. Suppose $u \notin C^*$. Then $u \in C$ for some other good component of ϕ, since by definition every vertex belongs to some good component, possibly of size 1. Now C induces a connected subgraph in $G - uv$, and all edges leaving C in $G - uv$ are bad with respect to ϕ by the definition of a good component. Since ϕ has less than $2k^2$ bad edges by Observation 4, it follows that $d_{G-uv}(C) < 2k^2$, and thus $d_G(C) \leq 2k^2$. Furthermore, because $|Y \cap C| \leq 2k^2$, we have that $d_{G^*}(C) \leq 4k^2$. Finally, by Observation 5, C must contain a vertex $x \in T_1 \cup T_2$. Hence there must be an (x, y^*)-important set X such that $C \subseteq X$ and $d(X) \leq 4k^2$ in G^*. But $C \subseteq X \subseteq Z$, which implies that $u \in Z$, contradicting the assumption that $u \notin Z$. The proof that $v \in C^*$ is identical. We conclude that both u and v belong to the good component C^*, and hence, by Observation 3, uv is irrelevant. ◀

▶ **Lemma 10.** *G contains an edge uv such that $u \notin Z$ and $v \notin Z$.*

Proof. We first prove that $d_G(Z) \leq (4k^2)\,t\,4^{4k^2}$ and $|Z \cap Y| \leq (4k^2)\,t\,4^{4k^2}$. For the first inequality, it suffices to show that $d_{G^*}(Z) \leq (4k^2)\,t\,4^{4k^2}$, because G is a subgraph of G^*. By Theorem 8, there exist at most 4^{4k^2} (x, y^*)-important sets with $d_{G^*}(X) \leq 4k^2$ for each $x \in T_1 \cup T_2$. Since Z is the union of all those sets over all elements of $T_1 \cup T_2$, and $|T_1 \cup T_2| \leq t$, the first inequality follows. To see that $|Z \cap Y| \leq (4k^2)\,t\,4^{4k^2}$, observe that each (x, y^*)-important set X in G^* with $d_{G^*}(X) \leq 4k^2$ contains at most $4k^2$ vertices of Y, since each vertex in Y is a neighbour of y^*.

In order to prove that G contains an edge uv with $u \notin Z$ and $v \notin Z$, we arbitrarily choose two disjoint subsets Y_1, Y_2 of Y such that $Z \cap Y \subseteq Y_2$ and $|Y_1| = |Y_2| = (4k^2)\,t\,4^{4k^2} + 1$. Since $|Y| \geq 2(4k^2)\,t\,4^{4k^2} + 2$ by assumption and we showed that $|Z \cap Y| \leq (4k^2)\,t\,4^{4k^2}$, such sets Y_1, Y_2 always exist. By the definition of a well-connected set, there are $|Y_1| = (4k^2)\,t\,4^{4k^2} + 1$ vertex-disjoint paths starting in Y_1 and ending Y_2. For every $1 \leq i \leq |Y_1|$, let $u_i v_i$ be the first edge on the ith such path, with $u_i \in Y_1$. Recall that $Z \cap Y \subseteq Y_2$ by assumption. Since all of the u_i's are in Y_1, none of them are in Z. Thus, each v_i that belongs to Z contributes one to $d_G(Z)$, as then $u_i v_i \in \delta_G(Z)$. Since we bounded $d_G(Z)$ from above by $(4k^2)\,t\,4^{4k^2}$ at the start of this proof, not every v_i can belong to Z. Hence there is an edge $u_i v_i$ with neither endpoint in Z. ◀

We are now ready to state the main lemma of this section.

▶ **Lemma 11.** Cheap Coloring Extension *can be solved in time $f(k, t)\,n^{O(1)}$.*

Proof. Let (G, k, t, T_1, T_2) be an instance of Cheap Coloring Extension, and let f be an appropriate function that does not depend on n. We first apply Theorem 4 and the remark immediately following it to compute, in time $f(k, t)\,n^{O(1)}$, either a tree-decomposition of G of width at most $3(4k^2)\,t\,4^{4k^2} + 3$ or a well-connected set Y of size at least $2(4k^2)\,t\,4^{4k^2} + 2$.

If we get a tree-decomposition of small width, we apply Lemma 6 to solve the problem in additional time $f(k,t)\,n$. If we find a well-connected set Y, we continue to find an irrelevant edge $e \in E(G)$ and delete it from G. Lemmas 9 and 10 guarantee that such an edge always exists. In order to find e, we first compute the set Z. We already argued that this can be done in time $f(k,t)\,n^{O(1)}$ if G and Y are given. We can find an irrelevant edge as explained in the proof of Lemma 10 in additional polynomial time, since this amounts to computing $Z \cap Y$, choosing Y_2 to contain the whole intersection and as many more vertices as needed to obtain $|Y_2| = (4k^2)\,t\,4^{4k^2}+1$, choosing Y_1 to be any subset of $Y \setminus Y_2$ such that $|Y_1| = |Y_2|$, and checking all edges leaving Y_1 to find one whose endpoints do not belong to Z. The total running time of this whole procedure is clearly $f(k,t)\,n^{O(1)}$.

After an irrelevant edge e is deleted from G, we run the whole procedure on $(G - e, k, t, T_1, T_2)$. This can be repeated at most $|E(G)| = n^{O(1)}$ times, and hence the total running time $f(k,t)\,n^{O(1)}$ follows. ◀

Our main result immediately follows from Lemmas 1, 2, 3, and 11.

▶ **Theorem 12.** Bipartite Contraction *is fixed parameter tractable when parameterized by* k.

We end this section with a remark on the running time. If we use Lemma 6 in the proof of Lemma 11, then the parameter dependence of the whole algorithm is dominated by a very large function in k [3]. However, as we remarked after Lemma 6, we can obtain a running time of $(4^{O(k^2)})^{4^{O(k^2)}}n = 2^{2^{O(k^2)}}n$ for the small treewidth case in the proof of Lemma 11. This is because the treewidth of the instance at hand is $4^{4k^2}k^{O(1)} = 4^{4k^2+O(\log k)} = 4^{O(k^2)}$ when the small treewidth case applies. This gives a total running time of of $2^{2^{O(k^2)}}n^{O(1)}$ for our algorithm for Bipartite Contraction.

5 Concluding Remarks

We showed that Bipartite Contraction is fixed parameter tractable. A highly relevant question is whether this problem admits a *polynomial kernel*, i.e., a polynomial time algorithm that transforms an instance (G,k) into an equivalent instance (G',k') of size $g(k)$, where g is a polynomial in k. Very recently, Kratsch and Wahlström [17] announced that the problems OCT and Edge Bipartization admit *randomized* polynomial kernels, which can be obtained using a novel kernelization approach based on matroid theory.

We conclude with the following question: Can some of the algorithms that currently use Robertson-Seymour machinery to find an irrelevant vertex, be modified in such a way that they find an irrelevant vertex using p-connected sets instead?

References

1 T. Asano and T. Hirata. Edge-contraction problems. *Journal of Computer and System Sciences*, 26:197–208, 1983.

2 A. E. Brouwer and H. J. Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11:71–79, 1987.

3 B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

4 J. Chen, Y. Liu, and S. Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.

5 J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5), 2008.

6 E. Demaine, M. Hajiaghayi and K. Kawarabayashi. Decomposition, approximation, and coloring of odd-minor-free graphs. In *Proceedings of SODA 2010*, 329–344, ACM-SIAM.

7 R. Diestel, K. Yu. Gorbunov, T. R. Jensen and C. Thomassen. Highly connected sets and the excluded grid theorem. *Journal of Combinatorial Theory, Series B*, 75:61–73, 1999.

8 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science, Springer-Verlag, 1999.

9 J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72:1386–1396, 2006.

10 F. Hüffner. Algorithm engineering for optimal graph bipartization. *Journal of Graph Algorithms and Applications*, 13(2):77–98, 2009.

11 R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 85–103, Plenum Press, New York, 1972.

12 K. Kawarabayashi. Planarity allowing few error vertices in linear time. In *Proceedings of FOCS 2009*, 639–648, IEEE.

13 K. Kawarabayashi. An improved algorithm for finding cycles through elements. In *Proceedings of IPCO 2008*, LNCS 5035:374–384, Springer.

14 K. Kawarabayashi and B. A. Reed. An (almost) linear time algorithm for odd cycles transversal. In *Proceedings of SODA 2010*, 365–378, ACM-SIAM.

15 K. Kawarabayashi and B. A. Reed. Odd cycle packing. In *Proceedings of STOC 2010*, 695–704, ACM.

16 K. Kawarabayashi and P. Wollan. A simpler algorithm and shorter proof for the graph minor decomposition. In *Proceedings of STOC 2010*, 687–694, ACM.

17 S. Kratsch and M. Wahlström. Compression via matroids: a randomized polynomial kernel for Odd Cycle Transversal. Manuscript available as *CoRR abs/1107.3068*.

18 A. Levin, D. Paulusma, and G. J. Woeginger. The computational complexity of graph contractions I: polynomially solvable and NP-complete cases. *Networks*, 51:178–189, 2008.

19 D. Lokshtanov and D. Marx Clustering with local restrictions. In *Proceedings of ICALP 2011*, LNCS 6755:785–797, Springer.

20 D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

21 D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proceedings of STOC 2011*, 469–478, ACM.

22 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

23 I. Razgon and B. O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75(8):435–450, 2009.

24 B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32:299–301, 2004.

25 N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

26 N. Robertson and P. D. Seymour. Graph Minors. XXII. Irrelevant vertices in linkage problems. Submitted.

27 T. Watanabe, T. Ae, and A. Nakamura. On the removal of forbidden graphs by edge-deletion or edge-contraction. *Discrete Applied Mathematics*, 3:151–153, 1981.

28 T. Watanabe, T. Ae, and A. Nakamura. On the NP-hardness of edge-deletion and edge-contraction problems. *Discrete Applied Mathematics*, 6:63–78, 1983.

29 S. Wernicke. *On the algorithmic tractability of single nucleotide polymorphism (SNP) analysis and related problems*. Diplomarbeit, WSI für Informatik, Universität Tübingen, 2003.

Simultaneously Satisfying Linear Equations Over $\mathbb{F}_2$: MaxLin2 and Max-r-Lin2 Parameterized Above Average

Robert Crowston[1], Michael Fellows[2], Gregory Gutin[1], Mark Jones[1], Frances Rosamond[2], Stéphan Thomassé[3], and Anders Yeo[1]

1 **Royal Holloway, University of London**
Egham, Surrey TW20 0EX, UK
2 **Charles Darwin University**
Darwin, Northern Territory 0909 Australia
3 **LIRMM-Université Montpellier II**
34392 Montpellier Cedex, France

Abstract

In the parameterized problem MaxLin2-AA$[k]$, we are given a system with variables $x_1, \ldots, x_n$ consisting of equations of the form $\prod_{i \in I} x_i = b$, where $x_i, b \in \{-1, 1\}$ and $I \subseteq [n]$, each equation has a positive integral weight, and we are to decide whether it is possible to simultaneously satisfy equations of total weight at least $W/2 + k$, where W is the total weight of all equations and k is the parameter (if $k = 0$, the possibility is assured). We show that MaxLin2-AA$[k]$ has a kernel with at most $O(k^2 \log k)$ variables and can be solved in time $2^{O(k \log k)}(nm)^{O(1)}$. This solves an open problem of Mahajan *et al.* (2006).

The problem Max-r-Lin2-AA$[k, r]$ is the same as MaxLin2-AA$[k]$ with two differences: each equation has at most r variables and r is the second parameter. We prove a theorem on Max-r-Lin2-AA$[k, r]$ which implies that Max-r-Lin2-AA$[k, r]$ has a kernel with at most $(2k - 1)r$ variables, improving a number of results including one by Kim and Williams (2010). The theorem also implies a lower bound on the maximum of a function $f : \{-1, 1\}^n \to \mathbb{R}$ whose Fourier expansion (which is a multilinear polynomial) is of degree r. We show applicability of the lower bound by giving a new proof of the Edwards-Erdős bound (each connected graph on n vertices and m edges has a bipartite subgraph with at least $m/2 + (n-1)/4$ edges) and obtaining a generalization.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases MaxLin; fixed-parameter tractability; kernelization; pseudo-boolean functions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.229

1 Introduction

1.1 MaxLin2-AA and Max-r-Lin2-AA. While MaxSat and its special case Max-r-Sat have been widely studied in the literature on algorithms and complexity for many years, MaxLin2 and its special case Max-r-Lin2 are less well known, but Håstad [24] succinctly summarized the importance of these two problems by saying that they are "as basic as satisfiability." These problems provide important tools for the study of constraint satisfaction problems such as MaxSat and Max-r-Sat since constraint satisfaction problems can often be reduced to MaxLin2 or Max-r-Lin2, see, e.g., [1, 2, 10, 11, 24, 26]. Accordingly, in the last decade, MaxLin2 and Max-r-Lin2 have attracted significant attention in algorithmics.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 229–240

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The problem MaxLin2 can be stated as follows. We are given a system of m equations in variables $x_1, \ldots, x_n$, where each equation is $\prod_{i \in I_j} x_i = b_j$ and $x_i, b_j \in \{-1, 1\}$, $j = 1, \ldots, m$, and where each equation is assigned a positive integral weight w_j. We are required to find an assignment of values to the variables in order to maximize the total weight of the satisfied equations.

Let W be the sum of the weights of all equations in S and let $\mathrm{sat}(S)$ be the maximum total weight of equations that can be satisfied simultaneously. To see that $W/2$ is a tight lower bound on $\mathrm{sat}(S)$ choose assignments to the variables independently and uniformly at random. Then $W/2$ is the expected weight of satisfied equations (as the probability of each equation being satisfied is $1/2$) and thus $W/2$ is a lower bound; to see the tightness consider a system consisting of pairs of equations of the form $\prod_{i \in I} x_i = -1$, $\prod_{i \in I} x_i = 1$ of the same weight, for some non-empty sets $I \subseteq [n]$. This leads to the following decision problem:

MaxLin2-AA
Instance: A system S of equations $\prod_{i \in I_j} x_i = b_j$, where $x_i, b_j \in \{-1, 1\}$, $j = 1, \ldots, m$ and where each equation is assigned a positive integral weight w_j; and a nonnegative integer k.
Question: $\mathrm{sat}(S) \geq W/2 + k$?

The maximization version of MaxLin2-AA (maximize k for which the answer is Yes), has been studied in the literature on approximation algorithms, cf. [24, 25]. These two papers also studied the following important special case of MaxLin2-AA:

Max-r-Lin2-AA
Instance: A system S of equations $\prod_{i \in I_j} x_i = b_j$, where $x_i, b_j \in \{-1, 1\}$, $|I_j| \leq r$, $j = 1, \ldots, m$; equation j is assigned a positive integral weight w_j, and a nonnegative integer k.
Question: $\mathrm{sat}(S) \geq W/2 + k$?

Håstad [24] proved that, as a maximization problem, Max-r-Lin2-AA with any fixed $r \geq 3$ (and hence MaxLin2-AA) cannot be approximated within a constant factor c for any $c > 1$ unless P=NP (that is, the problem is not in APX unless P=NP). Håstad and Venkatesh [25] obtained some approximation algorithms for the two problems. In particular, they proved that for Max-r-Lin2-AA there exists a constant $c > 1$ and a randomized polynomial-time algorithm that, with probability at least $3/4$, outputs an assignment with an approximation ratio of at most $c^r\sqrt{m}$.

The problem MaxLin2-AA was first studied in the context of parameterized complexity by Mahajan *et al.* [28] who naturally took k as the parameter[1]. We will denote this parameterized problem by MaxLin2-AA$[k]$. Despite some progress [10, 11, 22], the complexity of MaxLin2-AA$[k]$ has remained prominently open in the research area of "parameterizing above guaranteed bounds" that has attracted much recent attention (cf. [1, 7, 10, 11, 22, 26, 28]) and that still poses well-known and longstanding open problems (e.g., how difficult is it to determine if a planar graph has an independent set of size at least $(n/4) + k$?). One can parameterize Max-r-Lin2-AA by k for any fixed r (denoted by Max-r-Lin2-AA$[k]$) or by both k and r (denoted by Max-r-Lin2-AA$[k, r]$)[2].

[1] We provide basic definitions on parameterized algorithms and complexity in Subsection 1.4 below.
[2] While in the preceding literature only MaxLin2-AA$[k]$ was considered, we introduce and study Max-r-Lin2-AA$[k, r]$ in the spirit of Multivariate Algorithmics as outlined by Fellows [18] and Niedermeier [30].

Define the *excess* for $x^0 = (x_1^0, \ldots, x_n^0) \in \{-1, 1\}^n$ over S to be

$$\varepsilon_S(x^0) = \sum_{j=1}^{m} c_j \prod_{i \in I_j} x_i^0, \quad \text{where } c_j = w_j b_j.$$

Note that $\varepsilon_S(x^0)$ is the total weight of equations satisfied by x^0 minus the total weight of equations falsified by x^0. The maximum possible value of $\varepsilon_S(x^0)$ is the *maximum excess* of S. Håstad and Venkatesh [25] initiated the study of the excess of a system of equations and further research on the topic was carried out by Crowston *et al.* [11] who concentrated on MaxLin2-AA. In this paper, we study the maximum excess for Max-r-Lin2-AA. Note that the excess is a *pseudo-boolean function* [9], i.e., a function that maps $\{-1, 1\}^n$ to the set of reals.

1.2 Main Results and Structure of the Paper. Roughly speaking, a kernelization is a polynomial-time algorithm that transforms every instance $\mathcal{I}$ of the parameterized decision problem under consideration into an equivalent instance (called a kernel) $\mathcal{I}'$ of the same problem such that both the size of $\mathcal{I}'$ and the value of its parameter are bounded from above by a function in the parameter of $\mathcal{I}$ only.

Hereafter, $O(1)$ will denote an arbitrary absolute constant.

The main results of this paper are Theorems 6 and 9 outlined below. In 2006 Mahajan *et al.* [28] introduced MaxLin2-AA$[k]$ and asked what is its complexity. We answer this question in Theorem 6 by showing that MaxLin2-AA$[k]$ admits a kernel with at most $O(k^2 \log k)$ variables. Note that, in our kernel, only the number of variables is bounded from above by a polynomial in k. For the number of equations, we obtain an upper bound exponential in k. These two results imply that MaxLin2-AA$[k]$ admits a kernel and, hence, is fixed-parameter tractable (see Section 1.4). The proof of Theorem 6 is based on the main result in [11] and on a new algorithm for MaxLin2-AA$[k]$ of complexity $n^{2k}(nm)^{O(1)}$. We also prove that MaxLin2-AA$[k]$ can be solved in time $2^{O(k \log k)}(nm)^{O(1)}$ (Corollary 7).

The other main result of this paper, Theorem 9, gives a sharp lower bound on the maximum excess for Max-r-Lin2-AA as follows. Let S be an irreducible system (i.e., a system that cannot be reduced using Rule 1 or 2 defined below) and suppose that each equation contains at most r variables. Let $n \geq (k-1)r + 1$ and let $w_{\min}$ be the minimum weight of an equation of S. Then, in time $m^{O(1)}$, we can find an assignment x^0 to variables of S such that $\varepsilon_S(x^0) \geq k \cdot w_{\min}$.

In Section 2, we give some reduction rules for Max-r-Lin2-AA, describe an algorithm $\mathcal{H}$ introduced by Crowston *et al.* [11] and give some properties of the maximum excess, irreducible systems and Algorithm $\mathcal{H}$. In Section 3, we prove Theorem 6 and Corollary 7. A key tool in our proof of Theorem 9 is a lemma on sum-free subsets in a set of vectors from $\mathbb{F}_2^n$. The lemma and Theorem 9 are proved in Section 4. We prove several corollaries of Theorem 9 in Section 5. The corollaries are on parameterized and approximation algorithms as well as on lower bounds for the maxima of pseudo-boolean functions and their applications in graph theory. Our results on parameterized algorithms improve a number of previously known results including those of Kim and Williams [26]. In Section 6, we discuss some recent results and open problems.

1.3 Corollaries of Theorem 9. The following results have been obtained for Max-r-Lin2-AA$[k]$ when r is fixed and for Max-r-Lin2-AA$[k, r]$. Gutin *et al.* [22] proved that Max-r-Lin2-AA$[k]$ is fixed-parameter tractable and, moreover, has a kernel with $n \leq m = O(k^2)$. This kernel is, in fact, a kernel of Max-r-Lin2-AA$[k, r]$ with $n \leq m = O(9^r k^2)$. This kernel for Max-r-Lin2-AA$[k]$ was improved by Crowston *et al.* [11], with respect to the number of

variables, to $n = O(k \log k)$. For MAX-r-LIN2-AA$[k]$, Kim and Williams [26] were the first to obtain a kernel with a linear number of variables, i.e., $n = O(k)$. This kernel is, in fact, a kernel with $n \leq r(r+1)k$ for MAX-r-LIN2-AA$[k, r]$. In this paper, we obtain a kernel with $n \leq (2k-1)r$ for MAX-r-LIN2-AA$[k, r]$. As an easy consequence of this result we show that the maximization problem MAX-r-LIN2-AA is in APX if restricted to $m = O(n)$ and the weight of each equation is bounded by a constant. This is in the sharp contrast with the fact mentioned above that for each $r \geq 3$, MAX-r-LIN2-AA is not in APX.

Fourier analysis of *pseudo-boolean functions*, i.e., functions $f : \{-1, 1\}^n \to \mathbb{R}$, has been used in many areas of computer science (cf. [1, 11, 31]). In Fourier analysis, the Boolean domain is often assumed to be $\{-1, 1\}^n$ rather than more usual $\{0, 1\}^n$ and we will follow this assumption in our paper. Here we use the following well-known and easy to prove fact (see, e.g., [31]): each function $f : \{-1, 1\}^n \to \mathbb{R}$ can be uniquely written as

$$f(x) = \hat{f}(\emptyset) + \sum_{I \in \mathcal{F}} \hat{f}(I) \prod_{i \in I} x_i. \tag{1}$$

where $\mathcal{F} \subseteq \{I : \emptyset \neq I \subseteq [n]\}$, $[n] = \{1, 2, \ldots, n\}$ and $\hat{f}(I)$ are non-zero reals. Formula (1) is the *Fourier expansion* of f and $\hat{f}(I)$ are the *Fourier coefficients* of f. The right hand side of (1) is a polynomial and the degree $\max\{|I| : I \in \mathcal{F}\}$ of this polynomial will be called the *degree* of f. Let A be a $(0, 1)$-matrix with n rows and $|\mathcal{F}|$ columns and with entries a_{ij} such that $a_{ij} = 1$ if and only if term j in (1) contains x_i.

In Section 5, we obtain the following lower bound on the maximum of a pseudo-boolean function f of degree r:

$$\max_x f(x) \geq \hat{f}(\emptyset) + \lfloor (\mathrm{rank}A + r - 1)/r \rfloor \cdot \min\{|\hat{f}(I)| : I \in \mathcal{F}\}, \tag{2}$$

where $\mathrm{rank}A$ is the rank of A over $\mathbb{F}_2$. (Note that since $\mathrm{rank}A$ does not depend on the order of the columns in A, we may order the terms in (1) arbitrarily.)

To demonstrate the combinatorial usefulness of (2), we apply it to obtain a short proof of the well-known lower bound of Edwards-Erdős on the maximum size of a bipartite subgraph in a graph (the MAX CUT problem). Erdős [16] conjectured and Edwards [15] proved that every connected graph with n vertices and m edges has a bipartite subgraph with at least $m/2 + (n-1)/4$ edges. For short graph-theoretical proofs, see, e.g., Bollobás and Scott [7] and Erdős *et al.* [17]. We consider the BALANCED SUBGRAPH problem [3] that generalizes MAX CUT and show that our proof of the Edwards-Erdős bound can be easily extended to BALANCED SUBGRAPH. By contrast, the graph-theoretical proofs of the Edwards-Erdős bound do not seem to be easily extendable to BALANCED SUBGRAPH.

1.4 Parameterized Complexity and (Bi)kernelization. A *parameterized problem* is a subset $L \subseteq \Sigma^* \times \mathbb{N}$ over a finite alphabet Σ. L is *fixed-parameter tractable* (FPT, for short) if membership of an instance (x, k) in $\Sigma^* \times \mathbb{N}$ can be decided in time $f(k)|x|^{O(1)}$, where f is a function of the parameter k only. If membership can be decided in time $|x|^{O(f(k))}$ then L belongs to the parameterized complexity class XP. It is known that FPT is a proper subset of XP [14]. Analogs of NP are provided by the classes of parameterized problems of the W[t] Hierarchy giving the tower: FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ XP. For the definition of the classes W[t], see, e.g., [14, 19].

Given a pair L, L' of parameterized problems, a *bikernelization from L to L'* is a polynomial-time algorithm that maps an instance (x, k) to an instance (x', k') (the *bikernel*) such that (i) $(x, k) \in L$ if and only if $(x', k') \in L'$, (ii) $k' \leq f(k)$, and (iii) $|x'| \leq g(k)$ for some functions f and g. The function $g(k)$ is called the *size* of the bikernel. The notion of a bikernelization was introduced in [1], where it was observed that a parameterized problem

L is fixed-parameter tractable if and only if it is decidable and admits a bikernelization to a parameterized problem L'. A *kernelization* of a parameterized problem L is simply a bikernelization from L to itself; the bikernel is the *kernel*, and $g(k)$ is the *size* of the kernel. Due to the importance of polynomial-time kernelization algorithms in applied multivariate algorithmics, low degree polynomial size kernels and bikernels are of considerable interest, and the subject has developed substantial theoretical depth, cf. [1, 4, 5, 6, 13, 19, 20, 21, 22].

The case of several parameters $k_1, \ldots, k_t$ can be reduced to the one parameter case by setting $k = k_1 + \cdots + k_t$, see, e.g., [13].

2 Maximum Excess, Irreducible Systems and Algorithm $\mathcal{H}$

Recall that an instance of MaxLin2-AA consists of a system S of equations $\prod_{i\in I_j} x_i = b_j$, $j \in [m]$, where $\emptyset \neq I_j \subseteq [n]$, $b_j \in \{-1, 1\}$, $x_i \in \{-1, 1\}$. An equation $\prod_{i\in I_j} x_i = b_j$ has an integral positive weight w_j. Recall that the excess for $x^0 = (x_1^0, \ldots, x_n^0) \in \{-1, 1\}^n$ over S is $\varepsilon_S(x^0) = \sum_{j=1}^m c_j \prod_{i\in I_j} x_i^0$, where $c_j = w_j b_j$. The excess $\varepsilon_S(x^0)$ is the total weight of equations satisfied by x^0 minus the total weight of equations falsified by x^0. The maximum possible value of $\varepsilon_S(x^0)$ is the maximum excess of S.

▶ Remark 1. Observe that the answer to MaxLin2-AA is Yes if and only if the maximum excess is at least $2k$.

▶ Remark 2. The excess $\varepsilon_S(x)$ is a pseudo-boolean function and its Fourier expression is $\varepsilon_S(x) = \sum_{j=1}^m c_j \prod_{i\in I_j} x_i$. Moreover, observe that every pseudo-boolean function $f(x) = \sum_{I\in\mathcal{F}} \hat{f}(I) \prod_{i\in I} x_i$ (where $\hat{f}(\emptyset) = 0$) is the excess over the system $\prod_{i\in I} x_i = b_I$, $I \in \mathcal{F}$, where $b_I = 1$ if $\hat{f}(I) > 0$ and $b_I = -1$ if $\hat{f}(I) < 0$, with weights $|\hat{f}(I)|$. Thus, studying the maximum excess over a MaxLin2-AA-system (with real weights) is equivalent to studying the maximum of a pseudo-boolean function.

Consider two reduction rules for MaxLin2 studied in [22]. Rule 1 was studied before in [25].

▶ Reduction Rule 1. If we have, for a subset I of $[n]$, an equation $\prod_{i\in I} x_i = b'_I$ with weight w'_I, and an equation $\prod_{i\in I} x_i = b''_I$ with weight w''_I, then we replace this pair by one of these equations with weight $w'_I + w''_I$ if $b'_I = b''_I$ and, otherwise, by the equation whose weight is bigger, modifying its new weight to be the difference of the two old ones. If the resulting weight is 0, we delete the equation from the system.

Hereafter, rank$A$ will denote the rank of A over $\mathbb{F}_2$.

▶ Reduction Rule 2. Let A be the matrix over $\mathbb{F}_2$ corresponding to the set of equations in S, such that $a_{ji} = 1$ if $i \in I_j$ and 0, otherwise. Let $t =$ rank$A$ and suppose columns $a^{i_1}, \ldots, a^{i_t}$ of A are linearly independent. Then delete all variables not in $\{x_{i_1}, \ldots, x_{i_t}\}$ from the equations of S.

▶ **Lemma 1.** *[22] Let S' be obtained from S by Rule 1 or 2. Then the maximum excess of S' is equal to the maximum excess of S. Moreover, S' can be obtained from S in time polynomial in n and m.*

If we cannot change a weighted system S using Rules 1 and 2, we call it *irreducible*.

▶ **Lemma 2.** *Let S' be a system obtained from S by first applying Rule 1 as long as possible and then Rule 2 as long as possible. Then S' is irreducible.*

Proof. Let S^* denote the system obtained from S by applying Rule 1 as long as possible. Without loss of generality, assume that $x_1 \notin \{x_{i_1}, \ldots, x_{i_t}\}$ (see the description of Rule 2) and thus Rule 2 removes x_1 from S^*. To prove the lemma it suffices to show that after x_1 removal no pair of equations has the same left hand side. Suppose that there is a pair of equations in S^* which has the same left hand side after x_1 removal; let $\prod_{i\in I'} x_i = b'$ and $\prod_{i\in I''} x_i = b''$ be such equations and let $I' = I'' \cup \{1\}$. Then the entries of the first column of A, a^1, corresponding to the pair of equations are 1 and 0, but in all the other columns of A the entries corresponding to the the pair of equations are either 1,1 or 0,0. Thus, a^1 is independent from all the other columns of A, a contradiction. ◀

Let S be an irreducible system of MaxLin2-AA. Consider the following algorithm introduced in [11]. We assume that, in the beginning, no equation or variable in S is marked.

Algorithm $\mathcal{H}$

While the system S is nonempty do the following:

1. Choose an equation $\prod_{i\in I} x_i = b$ and mark a variable x_l such that $l \in I$.
2. Mark this equation and delete it from the system.
3. Replace every equation $\prod_{i\in I'} x_i = b'$ in the system containing x_l by $\prod_{i\in I\Delta I'} x_i = bb'$, where $I\Delta I'$ is the symmetric difference of I and I' (the weight of the equation is unchanged).
4. Apply Reduction Rule 1 to the system.

The *maximum $\mathcal{H}$-excess* of S is the maximum possible total weight of equations marked by $\mathcal{H}$ for S taken over all possible choices in Step 1 of $\mathcal{H}$. The following lemma proved by Crowston *et al.* [11] indicates the potential power of $\mathcal{H}$.

▶ **Lemma 3.** *Let S be an irreducible system. Then the maximum excess of S equals its maximum $\mathcal{H}$-excess. Furthermore, for any set of equations marked by Algorithm $\mathcal{H}$, in polynomial time, we can find an assignment of excess at least the total weight of marked equations.*

3 MaxLin2-AA

The following two theorems provide a basis for proving Theorem 6, the main result of this section.

▶ **Theorem 4.** *There exists an $n^{2k}(nm)^{O(1)}$-time algorithm for* MaxLin2-AA*[k] that returns an assignment of excess of at least $2k$ if one exists, and returns* NO *otherwise.*

Proof. Suppose we have an instance $\mathcal{L}$ of MaxLin2-AA$[k]$ that is reduced by Rules 1 and 2, and that the maximum excess of $\mathcal{L}$ is at least $2k$. Let A be the matrix introduced in Rule 2. Pick n equations $e_1, \ldots, e_n$ such that their rows in A are linearly independent. An assignment of excess at least $2k$ must either satisfy one of these equations, or falsify them all. If they are all falsified, then the system of equations $\bar{e}_1, \ldots, \bar{e}_n$, where each $\bar{e}_i$ is e_i with the changed right hand side, has a unique solution, an assignment of values to $x_1, \ldots, x_n$. If this assignment does not give excess at least $2k$ for $\mathcal{L}$, then any assignment that leads to excess at least $2k$ must satisfy at least one of $e_1, \ldots, e_n$. Thus, by Lemma 3, algorithm $\mathcal{H}$ can mark one of these equations and achieve an excess of at least $2k$.

This gives us the following depth-bounded search tree. At each node N of the tree, reduce the system by Rules 1 and 2, and let n' be the number of variables in the reduced

system. Then find n' equations $e_1, \ldots, e_{n'}$ corresponding to linearly independent vectors. Find an assignment of values to $x_1, \ldots, x_{n'}$ that falsifies all of $e_1, \ldots, e_{n'}$. Check whether this assignment achieves excess of at least $2k - w^*$, where w^* is total weight of equations marked by $\mathcal{H}$ in all predecessors of N. If it does, then return the assignment and stop the algorithm. Otherwise, split into n' branches. In the i'th branch, run an iteration of $\mathcal{H}$ marking equation e_i. Then repeat this algorithm for each new node. Whenever the total weight of marked equations is at least $2k$, return the suitable assignment. Clearly, the algorithm will terminate without an assignment if the maximum excess of $\mathcal{L}$ is less than $2k$.

All the operations at each node take time $(nm)^{O(1)}$, and there are less than n^{2k+1} nodes in the search tree. Therefore this algorithm takes time $n^{2k}(nm)^{O(1)}$. ◀

▶ **Theorem 5.** *[11] Let S be an irreducible system of* MaxLin2-AA*[k] and let $k \geq 2$. If $k \leq m \leq 2^{n/(k-1)} - 2$, then the maximum excess of S is at least k. Moreover, we can find an assignment with excess of at least k in time $m^{O(1)}$.*

▶ **Theorem 6.** *The problem* MaxLin2-AA*[k] has a kernel with at most $O(k^2 \log k)$ variables.*

Proof. Let $\mathcal{L}$ be an instance of MaxLin2-AA$[k]$ and let S be the system of $\mathcal{L}$ with m equations and n variables. We may assume that S is irreducible. Let the parameter k be an arbitrary positive integer.

If $m < 2k$ then $n < 2k = O(k^2 \log k)$. If $2k \leq m \leq 2^{n/(2k-1)} - 2$ then, by Theorem 5 and Remark 1, the answer to $\mathcal{L}$ is YES and the corresponding assignment can be found in polynomial time. If $m \geq n^{2k}$ then, by Theorem 4, we can solve $\mathcal{L}$ in polynomial time.

Finally we consider the case $2^{n/(2k-1)} - 1 \leq m \leq n^{2k} - 1$. Hence, $n^{2k} \geq 2^{n/(2k-1)}$. Therefore, $4k^2 \geq 2 + n/\log n \geq \sqrt{n}$ and $n \leq (2k)^4$. Hence, $n \leq 4k^2 \log n \leq 4k^2 \log(16k^4) = O(k^2 \log k)$.

Since S is irreducible, $m < 2^n$ and thus we have obtained the desired kernel. ◀

▶ **Corollary 7.** *The problem* MaxLin2-AA*[k] can be solved in time $2^{O(k \log k)}(nm)^{O(1)}$.*

Proof. Let $\mathcal{L}$ be an instance of MaxLin2-AA$[k]$. By Theorem 6, in time $(nm)^{O(1)}$ either we solve $\mathcal{L}$ or we obtain a kernel with at most $O(k^2 \log k)$ variables. In the second case, we can solve the reduced system (kernel) by the algorithm of Theorem 4 in time $[O(k^2 \log k)]^{2k} = [O(k^2 \log k)m]^{O(1)} = 2^{O(k \log k)} m^{O(1)}$. Thus, the total time is $2^{O(k \log k)}(nm)^{O(1)}$. ◀

4 Max-r-Lin2-AA

In order to prove Theorem 9, we will need the following lemma on vectors in $\mathbb{F}_2^n$. Let M be a set of m vectors in $\mathbb{F}_2^n$ and let A be a $m \times n$-matrix in which the vectors of M are rows. Using Gaussian elimination on A one can find a maximum size linearly independent subset of M in polynomial time [27]. Let K and M be sets of vectors in $\mathbb{F}_2^n$ such that $K \subseteq M$. We say K is *M-sum-free* if no sum of two or more distinct vectors in K is equal to a vector in M. Observe that K is M-sum-free if and only if K is linearly independent and no sum of vectors in K is equal to a vector in $M \backslash K$.

▶ **Lemma 8.** *Let M be a set of vectors in $\mathbb{F}_2^n$ such that M contains a basis of $\mathbb{F}_2^n$. Suppose that each vector of M contains at most r non-zero coordinates. If $k \geq 1$ is an integer and $n \geq r(k-1)+1$, then in time $|M|^{O(1)}$, we can find a subset K of M of k vectors such that K is M-sum-free.*

Proof. Let $\mathbf{1} = (1, \ldots, 1)$ be the vector in $\mathbb{F}_2^n$ in which every coordinate is 1. Note that $\mathbf{1} \notin M$. By our assumption M contains a basis of $\mathbb{F}_2^n$ and we may find such a basis in polynomial time (using Gaussian elimination, see above). We may write $\mathbf{1}$ as a sum of some vectors of this basis B. This implies that $\mathbf{1}$ can be expressed as follows: $\mathbf{1} = v_1 + v_2 + \cdots + v_s$, where $\{v_1, \ldots, v_s\} \subseteq B$ and $v_1, \ldots, v_s$ are linearly independent, and we can find such an expression in polynomial time.

For each $v \in M \backslash \{v_1, \ldots, v_s\}$, consider the set $S_v = \{v, v_1, \ldots, v_s\}$. In polynomial time, we may check whether S_v is linearly independent. Consider two cases:

Case 1: S_v is linearly independent for each $v \in M \backslash \{v_1, \ldots, v_s\}$. Then $\{v_1, \ldots, v_s\}$ is M-sum-free (here we also use the fact that $\{v_1, \ldots, v_s\}$ is linearly independent). Since each v_i has at most r positive coordinates, we have $sr \geq n > r(k-1)$. Hence, $s > k-1$ implying that $s \geq k$. Thus, $\{v_1, \ldots, v_k\}$ is the required set K.

Case 2: S_v is linearly dependent for some $v \in M \backslash \{v_1, \ldots, v_s\}$. Then we can find (in polynomial time) $I \subseteq [s]$ such that $v = \sum_{i \in I} v_i$. Thus, we have a shorter expression for $\mathbf{1}$: $\mathbf{1} = v'_1 + v'_2 + \cdots + v'_{s'}$, where $\{v'_1, \ldots, v'_{s'}\} = \{v\} \cup \{v_i : i \notin I\}$. Note that $\{v'_1, \ldots, v'_{s'}\}$ is linearly independent.

Since $s \leq n$ and Case 2 produces a shorter expression for $\mathbf{1}$, after at most n iterations of Case 2 we will arrive at Case 1. ◀

Now we can prove the main result of this section.

▶ **Theorem 9.** *Let S be an irreducible system and suppose that each equation contains at most r variables. Let $n \geq (k-1)r + 1$ and let $w_{\min}$ be the minimum weight of an equation of S. Then, in time $m^{O(1)}$, we can find an assignment x^0 to variables of S such that $\varepsilon_S(x^0) \geq k \cdot w_{\min}$.*

Proof. Consider a set M of vectors in $\mathbb{F}_2^n$ corresponding to equations in S as follows: for each equation $\prod_{i \in I} x_i = b$ in S, define a vector $v = (v_1, \ldots, v_n) \in M$, where $v_i = 1$ if $i \in I$ and $v_i = 0$, otherwise.

As S is reduced by Rule 2 we have that M contains a basis for $\mathbb{F}_2^n$, and each vector contains at most r non-zero coordinates and $n \geq (k-1)r + 1$. Therefore, using Lemma 8 we can find an M-sum-free set K of k vectors. Let $\{e_{j_1}, \ldots, e_{j_k}\}$ be the corresponding set of equations. Run Algorithm $\mathcal{H}$, choosing at Step 1 an equation of S from $\{e_{j_1}, \ldots, e_{j_k}\}$ each time, and let S' be the resulting system. Algorithm $\mathcal{H}$ will run for k iterations of the while loop as no equation from $\{e_{j_1}, \ldots, e_{j_k}\}$ will be deleted before it has been marked.

Indeed, suppose that this is not true. Then for some e_{j_l} and some other equation e in S, after applying Algorithm $\mathcal{H}$ for at most $l-1$ iterations e_{j_l} and e contain the same variables. Thus, there are vectors $v_j \in K$ and $v \in M$ and a pair of nonintersecting subsets K' and K'' of $K \setminus \{v, v_j\}$ such that $v_j + \sum_{u \in K'} u = v + \sum_{u \in K''} u$. Thus, $v = v_j + \sum_{u \in K' \cup K''} u$, contradicting the definition of K.

Thus, by Lemma 3, we are done. ◀

▶ Remark 3. To see that the inequality $n \geq r(k-1) + 1$ in the theorem is best possible assume that $n = r(k-1)$ and consider a partition of $[n]$ into $k-1$ subsets $N_1, \ldots, N_{k-1}$, each of size r. Let S be the system consisting of subsystems S_i, $i \in [k-1]$, such that a subsystem S_i is comprised of equations $\prod_{i \in I} x_i = -1$ of weight 1 for every I such that $\emptyset \neq I \subseteq N_i$. Now assume without loss of generality that $N_i = [r]$. Observe that the assignment $(x_1, \ldots, x_r) = (1, \ldots, 1)$ falsifies all equations of S_i but by setting $x_j = -1$ for any $j \in [r]$ we satisfy the equation $x_j = -1$ and turn the remaining equations into pairs

of the form $\prod_{i\in I} x_i = -1$ and $\prod_{i\in I} x_i = 1$. Thus, the maximum excess of S_i is 1 and the maximum excess of S is $k-1$.

▶ Remark 4. It is easy to check that Theorem 9 holds when the weights of equations in S are real numbers, not necessarily integers.

5 Applications of Theorem 9

▶ **Theorem 10.** *The problem* Max-r-Lin2-AA$[k,r]$ *has a kernel with at most* $(2k-1)r$ *variables.*

Proof. Let T be the system of an instance of Max-r-Lin2-AA$[k,r]$. After applying Rules 1 and 2 to T as long as possible, we obtain a new system S which is irreducible. Let n be the number of variables in S and observe that the number of variables in an equation in S is bounded by r (as in T). If $n \geq (2k-1)r+1$, then, by Theorem 9 and Remark 1, S is a Yes-instance of Max-r-Lin2-AA$[k,r]$ and, hence, by Lemma 1, S and T are both Yes-instances of Max-r-Lin2-AA$[k,r]$. Otherwise $n \leq (2k-1)r$ and since S is irreducible the number m of equations in S is less than 2^n. Thus, we have the required kernel. ◀

▶ **Corollary 11.** *The maximization problem* Max-r-Lin2-AA *is in APX if restricted to* $m = O(n)$ *and the weight of each equation bounded by a fixed constant.*

Proof. It follows from Theorem 9 and Remark 1 that the answer to Max-r-Lin2-AA, as a decision problem, is Yes as long as $2k \leq \lfloor (n+r-1)/r \rfloor$. This implies approximation ratio at most $W/(2\lfloor (n+r-1)/r \rfloor)$ which is bounded by a constant provided $m = O(n)$ and the weight of each equation is bounded by a constant (then $W = O(n)$). ◀

The (parameterized) Boolean Max-r-Constraint Satisfaction Problem (Max-r-CSP) generalizes MaxLin2-AA$[k,r]$ as follows: We are given a set Φ of Boolean functions, each involving at most r variables, and a collection $\mathcal{F}$ of m Boolean functions, each $f \in \mathcal{F}$ being a member of Φ, and each acting on some subset of the n Boolean variables $x_1, x_2, \ldots, x_n$ (each $x_i \in \{-1, 1\}$). We are to decide whether there is a truth assignment to the n variables such that the total number of satisfied functions is at least $E + k$, where E is the average value of the number of satisfied functions. The parameters are k and r.

Using a bikernelization algorithm described in [1, 11] and our new kernel result, it easy to see that Max-r-CSP with parameters k and r admits a bikernel with at most $(k2^{r+1}-1)r$ variables. This result improves the corresponding result of Kim and Williams [26] ($n \leq kr(r+1)2^r$).

The following result is essentially a corollary of Theorem 9 and Remark 4.

▶ **Theorem 12.** *Let*

$$f(x) = \hat{f}(\emptyset) + \sum_{I\in\mathcal{F}} \hat{f}(I) \prod_{i\in I} x_i \tag{3}$$

be a pseudo-boolean function of degree r*. Then*

$$\max_x f(x) \geq \hat{f}(\emptyset) + \lfloor (\mathrm{rank}A + r - 1)/r \rfloor \cdot \min\{|\hat{f}(I)| : I \in \mathcal{F}\}, \tag{4}$$

where A *is a* $(0,1)$*-matrix with entries* a_{ij} *such that* $a_{ij} = 1$ *if and only if term* j *in (3) contains* x_i *and* $\mathrm{rank}A$ *is the rank of* A *over* $\mathbb{F}_2$*. One can find an assignment of values to* x *satisfying (4) in time* $(n|\mathcal{F}|)^{O(1)}$.

Proof. By Remark 2 the function $f(x) - \hat{f}(\emptyset) = \sum_{I \in \mathcal{F}} \hat{f}(I) \prod_{i \in I} x_i$ is the excess over the system $\prod_{i \in I} x_i = b_I$, $I \in \mathcal{F}$, where $b_I = +1$ if $\hat{f}(I) > 0$ and $b_I = -1$ if $\hat{f}(I) < 0$, with weights $|\hat{f}(I)|$. Clearly, Rule 1 will not change the system. Using Rule 2 we can replace the system by an equivalent one (by Lemma 1) with rank$A$ variables. By Lemma 2, the new system is irreducible and we can now apply Theorem 9. By this theorem, Remark 2 and Remark 4, $\max_x f(x) \geq \hat{f}(\emptyset) + k^* \min\{|\hat{f}(I)| : I \in \mathcal{F}\}$, where k^* is the maximum value of k satisfying rank$A \geq (k-1)r + 1$. It remains to observe that $k^* = \lfloor (\text{rank}A + r - 1)/r \rfloor$. ◀

To give a new proof of the Edwards-Erdős bound, we need the following well-known and easy-to-prove fact [8]. For a graph $G = (V, E)$, an incidence matrix is a $(0,1)$-matrix with entries $m_{e,v}$, $e \in E$, $v \in V$ such that $m_{e,v} = 1$ if and only if v is incident to e.

▶ **Lemma 13.** *The rank over $\mathbb{F}_2$ of an incidence matrix M of a connected graph equals $|V| - 1$.*

▶ **Theorem 14.** *Let $G = (V, E)$ be a connected graph with n vertices and m edges. Then G contains a bipartite subgraph with at least $\frac{m}{2} + \frac{n-1}{4}$ edges. Such a subgraph can be found in polynomial time.*

Proof. Let $V = \{v_1, v_2, \ldots, v_n\}$ and let $c : V \to \{-1, 1\}$ be a 2-coloring of G. Observe that the maximum number of edges in a bipartite subgraph of G equals the maximum number of properly colored edges (i.e., edges whose end-vertices received different colors) over all 2-colorings of G. For an edge $e = v_i v_j \in E$ consider the following function $f_e(x) = \frac{1}{2}(1 - x_i x_j)$, where $x_i = c(v_i)$ and $x_j = c(v_j)$ and observe that $f_e(x) = 1$ if e is properly colored by c and $f_e(x) = 0$, otherwise. Thus, $f(x) = \sum_{e \in E} f_e(x)$ is the number of properly colored edges for c. We have $f(x) = \frac{m}{2} - \frac{1}{2} \sum_{e \in E} x_i x_j$. By Theorem 12, $\max_x f(x) \geq m/2 + \lfloor (\text{rank}A + 2 - 1)/2 \rfloor / 2$. Observe that matrix A in this bound is an incidence matrix of G and, thus, by Lemma 13 rank$A = n - 1$. Hence, $\max_x f(x) \geq \frac{m}{2} + \frac{1}{2} \lfloor \frac{n}{2} \rfloor \geq \frac{m}{2} + \frac{n-1}{4}$ as required. ◀

This theorem can be extended to the Balanced Subgraph problem [3], where we are given a graph $G = (V, E)$ in which each edge is labeled either by $=$ or by $\neq$ and we are asked to find a 2-coloring of V such that the maximum number of edges is satisfied; an edge labeled by $=$ ($\neq$, resp.) is *satisfied* if and only if the colors of its end-vertices are the same (different, resp.).

▶ **Theorem 15.** *Let $G = (V, E)$ be a connected graph with n vertices and m edges labeled by either $=$ or $\neq$. There is a 2-coloring of V that satisfies at least $\frac{m}{2} + \frac{n-1}{4}$ edges of G. Such a 2-coloring can be found in polynomial time.*

Proof. Let $V = \{v_1, v_2, \ldots, v_n\}$ and let $c : V \to \{-1, 1\}$ be a 2-coloring of G. Let $x_p = c(v_p)$, $p \in [n]$. For an edge $v_i v_j \in E$ we set $s_{ij} = 1$ if $v_i v_j$ is labeled by $\neq$ and $s_{ij} = -1$ if $v_i v_j$ is labeled by $=$. Then the function $\frac{1}{2} \sum_{v_i v_j \in E} (1 - s_{ij} x_i x_j)$ counts the number of edges satisfied by c. The rest of the proof is similar to that in the previous theorem. ◀

6 Open Problems

The kernels obtained in Theorems 6 and 10 are not of polynomial size as the number of equations in the kernels is not bounded by a polynomial in the parameter(s). The existence of polynomial-size kernels for MaxLin2-AA$[k]$ and Max-r-Lin2-AA$[k, r]$ remains an open question.

Perhaps the kernel obtained in Theorem 6 or the algorithm of Corollary 7 can be improved if we find a structural characterization of irreducible systems for which the maximum excess is less than $2k$. Such a characterization can be of interest by itself.

Let F be a CNF formula with clauses $C_1, \ldots, C_m$ of sizes $r_1, \ldots, r_m$. Since the probability of C_i being satisfied by a random assignment is $1 - 2^{-r_i}$, the expected (average) number of satisfied clauses is $E = \sum_{i=1}^{m}(1 - 2^{-r_i})$. It is natural to consider the following parameterized problem MaxSat-AA[k]: decide whether there is a truth assignment that satisfies at least $E+k$ clauses. When there is a constant r such that $|C_i| \leq r$ for each $i = 1, \ldots, m$, MaxSat-AA[k] is denoted by Max-r-Sat-AA[k]. Mahajan *et al.* [28] asked what is the complexity of Max-r-Sat-AA[k] and Alon *et al.* [1] proved that it is fixed-parameter tractable [1]. It would be interesting to determine the complexity[3] of MaxSat-AA[k].

In a graph $G = (V, E)$, a bisection (X, Y) is a partition of V into sets X and Y such that $|X| \leq |Y| \leq |X| + 1$. The size of (X, Y) is the number of edges between X and Y. In Max Bisection, we are given a graph G with $n \geq 2$ vertices and m edges and asked to find a bisection of maximum size. It is not hard to see that $\lceil m/2 \rceil$ is a tight lower bound on the maximum size of a bisection of G. Gutin and Yeo [23] proved that Max Bisection parameterized above $\lceil m/2 \rceil$ has a kernel with $O(k^2)$ vertices and $O(k^3)$ edges. Gutin and Yeo [23] also showed that $\lceil \frac{nm}{2(n-1)} \rceil$ is another tight lower bound on the maximum size of a bisection of G. Clearly, $\lceil \frac{nm}{2(n-1)} \rceil \leq \lceil m/2 \rceil$. Gutin and Yeo [23] left it as an open problem to determine the complexity of Max Bisection parameterized above $\lceil \frac{nm}{2(n-1)} \rceil$.

Finally, the entire area of parameterizing above or below tight guaranteed bounds offers many challenging open problems in parameterized complexity.

Acknowledgments The research of Crowston, Gutin, Jones and Yeo was partially supported by an International Joint grant of Royal Society. Fellows was supported by an Australian Research Council Professorial Fellowship. Gutin was also supported in part by the IST Programme of the European Community, under the PASCAL 2 Network of Excellence. The research of Rosamond was supported by an Australian Research Council Discovery Project. The research of Thomassé was partially supported by the AGAPE project (ANR-09-BLAN-0159).

References

1 N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo, Solving MAX-r-SAT above a tight lower bound. *Algorithmica* 61(3): 638–655 (2011).

2 N. Alon, G. Gutin and M. Krivelevich. Algorithms with large domination ratio, *J. Algorithms* 50: 118–131, 2004.

3 S. Böcker, F. Hüffner, A. Truss and M. Wahlström, A faster fixed-parameter approach to drawing binary tanglegrams. *IWPEC 2009*, Lect. Notes Comput. Sci. 5917: 38–49, 2009.

4 H. L. Bodlaender, R.G. Downey, M.R. Fellows, and D. Hermelin, On problems without polynomial kernels. *J. Comput. Syst. Sci.* 75(8): 423–434, 2009.

5 H.L. Bodlaender, B.M.P. Jansen and S. Kratsch, Cross-Composition: A new technique for kernelization lower bounds. *STACS 2011*: 165–176.

6 H. L. Bodlaender, S. Thomassé, and A. Yeo, Kernel bounds for disjoint cycles and disjoint paths. *ESA 2009*, Lect. Notes Comput. Sci. 5757: 635–646, 2009.

[3] Recently, Crowston *et al.* [12] determined the complexity of MaxSat-AA[k] by showing that MaxSat-AA[2] is NP-complete. Thus, MaxSat-AA[k] is not fixed-parameter tractable unless P=NP.

7 B. Bollobás and A. Scott, Better bounds for max cut. In *Contemporary Combinatorics* (B. Bollobás, ed.), Springer, 2002.
8 J.A. Bondy and U.S.R. Murty, *Graph Theory*. Springer, 2008.
9 E. Boros and P.L. Hammer, Pseudo-Boolean optimization. Discrete Applied Math. 123(1-3): 155–225, 2002.
10 R. Crowston, G. Gutin, and M. Jones, Note on Max Lin-2 above average. Inform. Proc. Lett. 110: 451–454, 2010.
11 R. Crowston, G. Gutin, M. Jones, E. J. Kim, and I. Ruzsa. Systems of linear equations over $\mathbb{F}_2$ and problems parameterized above average. *SWAT 2010*, Lect. Notes Comput. Sci. 6139 (2010), 164–175.
12 R. Crowston, G. Gutin, M. Jones, V. Raman, S. Saurabh, and A. Yeo, Parameterized Complexity of MaxSat Above Average. CoRR abs/1108.4501: (2011).
13 M. Dom, D. Lokshtanov and S. Saurabh, Incrompressibility though Colors and IDs, *ICALP 2009, Part I*, Lect. Notes Comput. Sci. 5555: 378–389, 2009.
14 R. G. Downey and M. R. Fellows. Parameterized Complexity, Springer, 1999.
15 C.S. Edwards, An improved lower bound for the number of edges in a largest bipartite subgraph. *Recent Advances in Graph Theory, Proc. 2nd Czecholslovac Symp.*, Academia, Prague, 1995, 167–181.
16 P. Erdős, On some extremal problems in graph theory. *Israel J. Math.* 3: 113–116, 1965.
17 P. Erdős, A. Gyárfás and Y. Kohayakawa, The size of the largest bipartite subgraphs. *Discrete Math.* 117: 267–271, 1997.
18 M. R. Fellows, Towards Fully Multivariate Algorithmics: Some New Results and Directions in Parameter Ecology. 20th International Workshop on Combinatorial Algorithms (IWOCA09), Lect. Notes Comput. Sci., 5874: 2–10, 2009.
19 J. Flum and M. Grohe. Parameterized Complexity Theory, Springer, 2006.
20 F.V. Fomin, D. Lokshtanov, N. Misra, G. Philip and S. Saurabh, Hitting forbidden minors: Approximation and kernelization. *STACS 2011*: 189–200.
21 G. Gutin, L. van Iersel, M. Mnich, and A. Yeo, All ternary permutation constraint satisfaction problems parameterized above average have kernels with quadratic number of vertices. *J. Comput. Syst. Sci.*, in press, doi:10.1016/j.jcss.2011.01.004.
22 G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. A probabilistic approach to problems parameterized above or below tight bounds. *J. Comput. Sys. Sci.* 77: 422–429, 2011.
23 G. Gutin and A. Yeo, Note on Maximal Bisection above Tight Lower Bound. *Inform. Proc. Lett.* 110: 966–969, 2010.
24 J. Håstad, Some optimal inapproximability results. *J. ACM* 48: 798–859, 2001.
25 J. Håstad and S. Venkatesh, On the advantage over a random assignment. *Random Structures & Algorithms* 25(2):117–149, 2004.
26 E.J. Kim and R. Williams, Improved Parameterized Algorithms for Constraint Satsfaction. Tech. Report arXiv:1008.0213, 2010.
27 B. Korte and J. Vygen, *Combinatorial Optimization: theory and algorithms*, 3rd Edition, Springer, 2006.
28 M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. J. Computer System Sciences, 75(2):137–153, 2009. A preliminary version appeared in the 2nd IWPEC, *Lect. Notes Comput. Sci.* 4169:38–49, 2006.
29 R. Niedermeier. Invitation to Fixed-Parameter Algorithms. Oxford University Press, 2006.
30 R. Niedermeier, Reflections on Multivariate Algorithmics and Problem Parameterization. STACS 2010: 17–32.
31 R. O'Donnell, Some topics in analysis of Boolean functions. Technical report, ECCC Report TR08-055, 2008. Paper for an invited talk at STOC'08, www.eccc.uni-trier.de/eccc-reports/2008/TR08-055/ .

Rainbow Connectivity: Hardness and Tractability

Prabhanjan Ananth[1], Meghana Nasre[1], and Kanthi K. Sarpatwar[2]

1 **Indian Institute of Science**
Bangalore
prabhanjan,meghana@csa.iisc.ernet.in
2 **University of Maryland**
College Park, Maryland, U.S.A.
kanthik@gmail.com

Abstract

A path in an edge colored graph is said to be a rainbow path if no two edges on the path have the same color. An edge colored graph is (strongly) rainbow connected if there exists a (geodesic) rainbow path between every pair of vertices. The (strong) rainbow connectivity of a graph G, denoted by ($src(G)$, respectively) $rc(G)$ is the smallest number of colors required to edge color the graph such that G is (strongly) rainbow connected. In this paper we study the rainbow connectivity problem and the strong rainbow connectivity problem from a computational point of view. Our main results can be summarised as below:

- For every fixed $k \geq 3$, it is NP-Complete to decide whether $src(G) \leq k$ even when the graph G is bipartite.
- For every fixed odd $k \geq 3$, it is NP-Complete to decide whether $rc(G) \leq k$. This resolves one of the open problems posed by Chakraborty et al. [4] hardness for the even case.
- The following problem is *fixed parameter tractable*: Given a graph G, determine the maximum number of pairs of vertices that can be rainbow connected using two colors.
- For a directed graph G, it is NP-Complete to decide whether $rc(G) \leq 2$.

1998 ACM Subject Classification G.2 Discrete Mathematics

Keywords and phrases Computational Complexity, Rainbow Connectivity, Graph Theory, Fixed Parameter Tractable Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.241

1 Introduction

This paper deals with the notion of *rainbow connectivity* and *strong rainbow connectivity* of a graph. Unless mentioned otherwise, all the graphs are assumed to be connected and undirected. Consider an edge coloring (not necessarily proper) of a graph $G = (V, E)$. A path between a pair of vertices is said to be a *rainbow path*, if no two edges on the path have the same color. If the edges of G can be colored using k colors such that, between every pair of vertices there exists a rainbow path then G is said to be k-rainbow connected. Further, if the k-coloring ensures that between every pair of vertices one of its geodesic *i.e.,* one of the shortest paths is a rainbow path, then G is said to be k-strongly rainbow connected. The minimum number of colors required to (strongly) rainbow connect a graph G is called the (strong) rainbow connection number denoted by ($src(G)$, respectively) $rc(G)$.

The concept of rainbow connectivity was recently introduced by Chartrand et al. in [6] as a measure of strengthening connectivity. The rainbow connection problem, apart from being an interesting combinatorial property, also finds an application in routing messages on cellular networks [4]. In their original paper [6], Chartrand et al. determined $rc(G)$

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 241–251

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and $src(G)$, in special cases where G is a complete bipartite or multipartite graph. Rainbow connectivity from a computational point of view was first studied by Caro et al. [3] who conjectured that computing the rainbow connection number of a given graph is NP-hard. This conjecture was confirmed by Chakraborty et al. [4], who proved that even deciding whether rainbow connection number of a graph equals 2 is NP-Complete. They further showed that the problem of deciding whether rainbow connection of a graph is at most k is NP-hard where k is an even integer. The status of the k-rainbow connectivity problem was left open for the case when k is odd. One of our results is to resolve this problem.

Our Results. We present the following new results in this paper:

1. For every fixed $k \geq 3$, deciding whether $src(G) \leq k$, is NP-Complete even when G is bipartite. As a consequence of our reduction, we show that it is NP-hard to approximate the problem of finding the strong connectivity of a graph by a factor of $n^{\frac{1}{2}-\epsilon}$, where n is the number of vertices in G.
2. For every fixed odd $k \geq 3$, deciding whether $rc(G) \leq k$ is NP-Complete.
3. We consider the following natural extension of the 2-rainbow connectivity problem: Given a graph G, determine the maximum number of pairs of vertices that can be rainbow connected with two colors. We show that the above problem is *fixed parameter tractable* when the number of pairs to be rainbow connected is a parameter.
4. We extend the notion of rainbow connectivity for directed graphs and show that for a directed graph G it is NP-Complete to decide whether $rc(G) \leq 2$.

In [4], Chakraborty et al. introduced the problem of *subset rainbow connectivity*, where in addition to the graph $G = (V, E)$ we are given a set P containing pairs of vertices. The goal is to answer whether there exists an edge coloring of G with k colors such that every pair in P has a rainbow path. We also use the subset rainbow connectivity problem and analogously define the subset strong rainbow connectivity problem to prove our hardness results.

Related Work. The problem of rainbow connectivity has received considerable attention after it was introduced by Chartrand et al. in [6]. Caro et al. [3], Krivelevich et al. [9], Chandran et al. [5] gave lower bounds for rainbow connection number of graphs as a function of the number of vertices and the minimum degree of the graph. Upper bounds were also given by Chandran et al. [5] for special graphs like interval graphs and AT-free graphs. In [2], Basavaraju et al. gave a constructive argument to show that any graph G can be colored with $r(r+2)$ colors in polynomial time where r is the radius of the graph. The threshold function for random graph to have $rc(G) = 2$ was studied by Caro et al. [3]. In case of strong rainbow connection number, Li et al. [10] and Li and Sun [11] gave upper bounds on some special graphs. Interestingly, no good upper bounds are known for the strong rainbow connection number in the general case.

2 Strong rainbow connectivity

In this section, we prove the hardness result for the k-strong rainbow connectivity problem: given a graph G and an integer $k \geq 3$, decide whether $src(G) \leq k$. In order to show the hardness of this problem, we first consider an intermediate problem called the *k-subset strong rainbow connectivity* problem. The input to the k-subset strong rainbow connectivity problem is a graph $G = (V, E)$ along with a set of pairs $P = \{(u, v) : (u, v) \subseteq V \times V\}$ and an integer k. Our goal is to answer whether there exists an edge coloring of G with at most k colors such that every pair $(u, v) \in P$ has a geodesic rainbow path.

The overall plan is to prove that k-subset strong rainbow connectivity is NP-hard by

showing a reduction from the vertex coloring problem. We then establish the polynomial time equivalence of the k-subset strong rainbow connectivity problem and the k-strong rainbow connectivity problem.

2.1 k-subset strong rainbow connectivity

Let $G = (V, E)$ be an instance of the k-vertex coloring problem. The problem is to decide if there exists an assignment of at most k colors to the vertices of G such that no pair of adjacent vertices are colored using the same color. This is one of the most well-studied problems in computer science and is known to be NP-hard for $k \geq 3$. Given an instance $G = (V, E)$ of the k-vertex coloring problem, we construct an instance $\langle G' = (V', E'), P\rangle$ of the k-subset strong rainbow connectivity problem.

The graph G' that we construct is a star, with one leaf vertex corresponding to every vertex $v \in V$ and an additional central vertex a. The set of pairs P captures the edges in E, that is, for every edge $(u, v) \in E$ we have a pair (u, v) in the set P. The goal is to color the edges of G' using at most k colors such that every pair in the set P has a geodesic rainbow path. More formally, we define the parameters $\langle G' = (V', E'), P\rangle$ of the k-subset strong rainbow connectivity problem below:

$$\begin{aligned} V' &= \{a\} \cup V \\ E' &= \{(a, v) : v \in V\} \\ P &= \{(u, v) : (u, v) \in E\} \end{aligned}$$

We now prove the following lemma which establishes the hardness of the k-subset strong rainbow connectivity problem.

▶ **Lemma 1.** *The graph $G = (V, E)$ is vertex colorable using $k(\geq 3)$ colors iff the graph $G' = (V', E')$ can be edge colored using k colors such that for every pair $(u, v) \in P$ there is a geodesic rainbow path between u and v in G'.*

Proof. Assume that G can be vertex colored using k colors; we show an assignment of colors to the edges of the graph G'. Let c be the color assigned to a vertex $v \in V$; we assign the color c to the edge $(a, v) \in E'$. Now consider any pair $(u, v) \in P$. Recall that $(u, v) \in P$ because there exists an edge $(u, v) \in E$. Since the coloring was a proper vertex coloring of G, the edges (a, u) and (a, v) in G' are assigned different colors by our coloring. Thus, the path $u - a - v$ is a rainbow path; further since that is the only path between u and v it is also a geodesic rainbow path.

To prove the other direction, assume that there exists an edge coloring of G' using k colors such that between every pair of vertices in P there is a geodesic rainbow path. It is easy to see that if we assign the color c of the edge $(a, v) \in E'$ to the vertex $v \in V$, we get a coloring that is a proper vertex coloring for G. ◀

Recall the problem of subset rainbow connectivity where we are content with any rainbow path between every pair in P. Note that our graph G' constructed in the above reduction is a tree, in fact a star and hence between every pair of vertices in P there is exactly one path. Thus, all the above arguments apply for the k-subset rainbow connectivity problem as well. As a consequence we can conclude the following:

▶ **Lemma 2.** *For every $k \geq 3$, both the problems k-subset strong rainbow connectivity and k-subset rainbow connectivity are NP-hard even when the input graph G is a star.*

2.2 k-strong rainbow connectivity

In this section, we establish the hardness of deciding whether a given graph can be strongly rainbow connected using k colors.

▶ **Theorem 3.** *For every $k \geq 3$, deciding whether a given graph G can be strongly rainbow colored using k colors is NP-hard. Further, the hardness holds even when the graph G is bipartite.*

Proof. We reduce the k-subset strong rainbow connectivity problem to the k-strong rainbow connectivity problem. Let $\langle G = (V, E), P\rangle$ be an instance of the k-subset strong rainbow connectivity problem. Using Lemma 2, we know that k-subset strong rainbow connectivity is NP-hard even when G is a star as well as the pairs $(v_i, v_j) \in P$ are such that both v_i and v_j are leaf nodes of the star. We assume both these properties on the input $\langle G, P\rangle$ and use them crucially in our reduction. Let us denote the central vertex of the star G by a and the leaf vertices by $L = \{v_1, \ldots, v_n\}$, that is, $V = \{a\} \cup L$. Using the graph G and the pairs P, we construct the new graph $G' = (V', E')$ as follows: for every leaf node $v_i \in L$, we introduce two new vertices u_i and u'_i. For every pair of leaf nodes $(v_i, v_j) \in (L \times L) \setminus P$, we introduce two new vertices $w_{i,j}$ and $w'_{i,j}$.

$$\begin{aligned} V' &= V \cup V_1 \cup V_2 \\ V_1 &= \{u_i : i \in \{1, \ldots, n\}\} \cup \{w_{i,j} : (v_i, v_j) \in (L \times L) \setminus P\} \\ V_2 &= \{u'_i : i \in \{1, \ldots, n\}\} \cup \{w'_{i,j} : (v_i, v_j) \in (L \times L) \setminus P\} \end{aligned}$$

The edge set E' is be defined as follows:

$$\begin{aligned} E' &= E \cup E_1 \cup E_2 \cup E_3 \\ E_1 &= \{(v_i, u_i) : v_i \in L, u_i \in V_1\} \cup \{(v_i, w_{i,j}), (v_j, w_{i,j}) : (v_i, v_j) \in (L \times L) \setminus P\} \\ E_2 &= \{(x, x') : x \in V_1, x' \in V_2\} \\ E_3 &= \{(a, x') : x' \in V_2\} \end{aligned}$$

We now prove that G' is k-strong rainbow connected iff $\langle G, P\rangle$ is k-subset strong rainbow connected. To prove one direction, we first note that, for all pairs $(v_i, v_j) \in P$, there is a two length path $v_i - a - v_j$ in G and this path is also present in G'. Further, this path is the only two length path in G' between v_i and v_j; hence any strong rainbow coloring of G' using k colors must make this path a rainbow path. This implies that if G cannot be edge colored with k colors such that every pair in P is strongly rainbow connected, the graph G' cannot be strongly rainbow colored using k colors.

To prove the other direction, assume that there is an edge coloring $\chi : E \to \{c_1, c_2, \ldots, c_k\}$ of G such that all pairs in P are strongly rainbow connected. We extend this edge coloring of G to an edge coloring of G', denoted by χ', such that G' is strong rainbow connected:

- We retain the color on the edges of G, *i.e.* $\chi'(e) = \chi(e) : e \in E$.
- For each edge $(v_i, u_i) \in E_1$, we set $\chi'(v_i, u_i) = c_3$.
- For each pair of edges $\{(v_i, w_{i,j}), (v_j, w_{i,j})\} \in E_1$, we set $\chi'(v_i, w_{i,j}) = c_1$, $\chi'(v_i, w_{i,j}) = c_2$ (Assume without loss of generality that $i < j$).
- The edges in E_2 form a complete bipartite graph between the vertices in V_1 and V_2. To color these edges, we pick a perfect matching M of size $|V_1|$ and assign $\chi'(e) = c_1, \forall e \in E_2 \cap M$ and $\chi'(e) = c_2, \forall e \in E_2 \setminus M$.
- Finally, for each edge $(a, x') \in E_3$, we set $\chi'(a, x') = c_3$.

It is straightforward to verify that this coloring makes G' strong rainbow connected. This completes the proof of NP-hardness of the k-strong rainbow connectivity problem.

We further note that the graph G' constructed above is in fact bipartite. The vertex set V' can be partitioned into two sets A and B, where $A = \{a\} \cup V_1$ and $B = L \cup V_2$ such that there are no edges between vertices in the same partition. This establishes the fact that the k-strong rainbow connectivity problem is NP-hard even for the bipartite case. ◀

From the same construction when $k = 3$, it follows that deciding whether a given graph G can be rainbow colored using at most 3 colors is NP-hard. To see this, note that between any pair of vertices $(v_i, v_j) \in P$, a path in G' that is not contained in G is of length at least 4 and the shortest path between v_i and v_j is in G. Further, we always color the edges $E' \setminus E$ using 3 colors; hence none of these paths can be rainbow path. Thus, we conclude the following corollary.

▶ **Corollary 4.** *Deciding whether $rc(G) \leq 3$ is NP-hard even when the graph G is bipartite.*

As a consequence of the reduction from the k-subset strong rainbow connectivity to the k-strong rainbow connectivity, we have the following result (the proof is in full version [1]):

▶ **Theorem 5.** *There is no polynomial time algorithm that approximates strong rainbow connection number of a graph $G = (V, E)$ within a factor of $n^{\frac{1}{2}-\epsilon}$, unless $NP = ZPP$. Here n denotes the number of vertices of G.*

3 Rainbow connectivity

In this section we investigate the complexity of deciding whether the rainbow connection number of a graph is at most k. We prove the NP-hardness of the k-rainbow connectivity problem i.e., deciding whether $rc(G) \leq k$, when k is odd. We recall from Lemma 2 that the k-subset rainbow connectivity problem is NP-hard. In the following theorem, we give a reduction of the k-subset rainbow connectivity problem to the k-rainbow connectivity problem.

▶ **Theorem 6.** *For every odd integer $k \geq 3$, deciding whether $rc(G) \leq k$ is NP-Complete.*

Proof. Let $\langle G = (V, E), P\rangle$ be an instance of the k-subset rainbow connectivity problem. Since k is assumed to be odd, let $k = 2m + 1$ where $m \in \mathbb{N}$. Let us denote the vertices of G as $V = \{v_1, \ldots, v_n\}$. Given the graph G and a set of pairs of vertices P, we construct an instance $G' = (V', E')$ of the k-rainbow connectivity problem as follows: For each vertex $v_i \in V$, we add $2m$ new vertices denoted by $u_{i,j}$ where $j \in \{1, \ldots, 2m\}$. Further, we add the following two paths: $v_i - u_{i,1} - u_{i,2} \cdots - u_{i,m}$ and $v_i - u_{i,m+1} - u_{i,m+2} \cdots - u_{i,2m}$. We also add edges $(u_{i,m}, u_{i,2m})$ and $(u_{i,1}, u_{i,m+1})$ (if $m = 1$, we only add one edge). For every pair of vertices $(v_i, v_j) \in (V \times V) \setminus P$: we add the edges $(u_{i,m}, u_{j,2m})$ and $(u_{i,2m}, u_{j,m})$. We add two more new vertices x, y and for every $v_i \in V$ we add the following edges: $(x, u_{i,m})$, $(x, u_{i,2m})$, $(y, u_{i,m})$ and $(y, u_{i,2m})$. Figure 1 shows a subgraph of the graph G'. The construction shows extra vertices added corresponding to v_i and v_j such that the pair $(v_i, v_j) \in (V \times V) \setminus P$. More formally, the vertex set V' can be defined as:

$$\begin{aligned} V' &= V \cup V_{1,m} \cup V_{m+1,2m} \cup V_{x,y} \\ V_{1,m} &= \{u_{i,j} : v_i \in V, j \in \{1, \ldots, m\}\} \\ V_{m+1,2m} &= \{u_{i,j} : v_i \in V, j \in \{m+1, \ldots, 2m\}\} \\ V_{x,y} &= \{x, y\} \end{aligned}$$

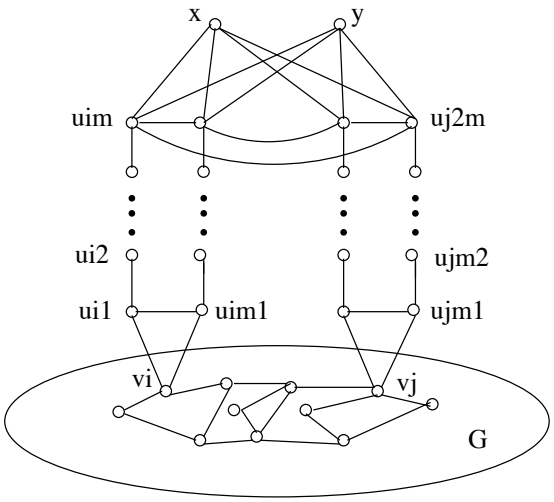


Figure 1 A subgraph of the graph G'. The construction shows extra vertices added corresponding to vertices v_i and v_j belonging to G. The pair $(v_i, v_j) \in (V \times V) \setminus P$.

The edge set E' can be defined as:

$$\begin{aligned}
E' &= E \cup E_1 \cup E_2 \cup E_{x,y} \\
E_1 &= \{(u_{i,j}, u_{i,j+1}) \ : \ v_i \in V, \ j \in \{1, \ldots, m\}, \ j \pmod m \neq 0\} \cup \\
&\quad \{(u_{i,1}, u_{i,m+1}), (u_{i,m}, u_{i,2m}) : v_i \in V\} \\
E_2 &= \{(u_{i,m}, u_{j,2m}), (u_{i,2m}, u_{j,m}) \ : \ (v_i, v_j) \in (V \times V) \setminus P\} \cup \\
&= \{(v_i, u_{i,1}), (v_i, u_{i,m+1}) : v_i \in V\} \\
E_{x,y} &= \{(x, u_{i,m}), (x, u_{i,2m}), (y, u_{i,m}), (y, u_{i,2m}) \ : \ i \in \{1, \ldots, n\}\}
\end{aligned}$$

We claim that G can be edge colored using k colors such that every pair belonging to P is rainbow connected if and only if $rc(G') \leq k$. Assume that G can be edge colored using k colors such that all pairs in P are rainbow connected. Let $\chi : E \to \{c_1, \ldots, c_{2m+1}\}$ be such a coloring. We obtain a coloring $\chi' : E' \to \{c_1, \ldots, c_{2m+1}\}$ as follows:

- For every $v_i \in V$:
  $\chi'(v_i, u_{i,1}) = c_1$; $\chi'(v_i, u_{i,m+1}) = c_{m+1}$;
  $\chi'(u_{i,j}, u_{i,j+1}) = c_{j+1}$ where $j \in \{1, \ldots, 2m-1\}$ and $j \pmod m \neq 0$;
  $\chi'(x, u_{i,m}) = c_{m+1}$; $\chi'(x, u_{i,2m}) = c_{2m+1}$;
  $\chi'(y, u_{i,m}) = c_{2m+1}$; $\chi'(y, u_{i,2m}) = c_1$.
  If $m \neq 1$, $\chi'(u_{i,1}, u_{i,m+1}) = c_{m+1}$; $\chi'(u_{i,m}, u_{i,2m}) = c_1$
  else $\chi'(u_{i,1}, u_{i,m+1}) = c_1$.
- For every $(v_i, v_j) \in (V \times V) \setminus P$: $\chi'(u_{i,m}, u_{j,2m}) = c_{2m+1}$ and $\chi'(u_{i,2m}, u_{i,m}) = c_{2m+1}$.
- For every edge $(v_i, v_j) \in E$: $\chi'(v_i, v_j) = \chi(v_i, v_j)$.

We claim that if χ makes G k-subset rainbow connected then χ' makes the graph G' k-rainbow connected. This requires a case-wise analysis, which we defer to the full version [1].

To prove the other direction, assume that $rc(G') \leq k$. Let $\chi : E' \to \{c_1, \ldots, c_k\}$ be an edge coloring of G' such that χ makes G' rainbow connected. We will translate this edge coloring of G' to an edge coloring of G as follows: color the edge (v_i, v_j) in G with the color $\chi(v_i, v_j)$. We claim that all pairs in P are rainbow connected in G. This follows from the observation that for a pair $(v_i, v_j) \in P$, any path between v_i and v_j which is of length at most

$2m+1$ in G' has to be completely contained in G. Since χ makes G' rainbow connected, the rainbow path between v_i and v_j in G' has to lie completely inside G itself. Correspondingly, there is a rainbow path between v_i and v_j in G. Hence, all pairs in P are rainbow connected in G. This proves that k-rainbow connectivity problem is NP-hard.

It is clear that given an edge k-coloring, for $k \in \mathbb{N}$, we can check in polynomial time, that the edge coloring rainbow connects every pair of vertices. Hence the problem of deciding if $rc(G) \leq k$ is in NP. The result follows. ◀

Unlike the case of strong rainbow connectivity, the reduction presented above does not give any insight into the inapproximability of the problem of finding the rainbow connection number of a graph. The reason being that the reduction stated in the proof of Theorem 3 yields an instance of k-strong rainbow connectivity problem which is independent of k *i.e.,* the structure of the graph does not change with k. On the contrary, the size of the instance of k-rainbow connectivity problem obtained from the reduction in Theorem 6 crucially depends on k.

3.1 Parameterized complexity

In this section, we study the parameterized complexity of a maximization version of the rainbow connectivity problem. Before that, we describe the necessary preliminaries. A problem is said to be fixed parameter tractable (FPT) with respect to a parameter k^*, if given an instance x of size $|x|$ there exists an algorithm with running time $f(k) \times |x|^{O(1)}$ where f is a function of k which is independent of $|x|$. One way of showing that a problem is fixed parameter tractable is to exhibit polynomial time reductions to obtain a *kernel* which is basically an equivalent instance whose size is purely a function of the parameter k. If the size of the kernel is a linear function in k then the kernel is termed as a *linear kernel*. For formal definitions, we refer the reader to [7, 8].

We are interested in the following problem: Given a graph $G = (V, E)$, color the edges of G using 2 colors such that maximum number of pairs are rainbow connected. Since deciding whether $rc(G) \leq 2$ is NP-Complete [4], it follows that the above maximization problem is NP-hard. Any edge coloring of a graph $G = (V, E)$ with 2 colors, trivially satisfies $|E|$ pairs. Hence, we are interested in deciding whether G can be 2-colored such that at least $|E| + k$ pairs of vertices are rainbow connected, where k is a parameter. We show that the problem is *fixed parameter tractable* with respect to k.

We first state a useful lemma (proof in full version [1]). Let us call a *non-edge* in G as an anti-edge; formally we call $e = (u, v)$ an anti-edge of a graph $G = (V, E)$ if $e \notin E$.

▶ **Lemma 7.** *Let $G = (V, E)$ be a connected graph with at least k anti-edges and a clique of size $\geq k$. The edges of G can be colored using 2 colors such that at least $|E| + k$ pair of vertices are rainbow-connected.*

Using the above lemma 7 we now show that the problem is fixed parameter tractable.

▶ **Theorem 8.** *Given a graph $G = (V, E)$, decide whether the edges of G can be colored using 2 colors such that at least $|E| + k$ pair of vertices are rainbow connected. The above problem has a kernel with at most $4k$ vertices and hence is fixed parameter tractable.*

* A parameter is a natural number associated to a problem instance. For example, a parameter could be the number of vertices of a graph instance in a vertex cover problem or the number of processors in a scheduling problem.

Proof. Let v be any arbitrary vertex and let O_v be the set of vertices which are not adjacent to v. We claim that there is a coloring which rainbow connects at least $|O_v|$ pair of non-adjacent vertices. Consider a breadth first search (bfs) tree rooted at v. Denote the set of vertices in each level of the bfs tree by L_i, $i \geq 1$. Then, $L_1 = \{v\}$, $L_2 = N(v)$ and $O_v = \cup_{i>2} L_i$. We now color the edges from L_{i-1} to L_i by red if i is odd and by blue if i is even. For $i > 2$, every vertex of L_i is rainbow connected to some vertex of L_{i-2}. Thus we have $|O_v|$ pairs of non-adjacent vertices rainbow connected by this coloring. Hence if $|O_v| \geq k$ for any vertex $v \in V$, we have a trivial *yes* instance at hand. Otherwise, $|O_v| < k$, for all $v \in V$.

Recall that our goal is to color the graph using 2 colors such that at least $|E| + k$ pair of vertices are rainbow connected. If G has less than k anti-edges, clearly this is not possible and we have a *no* instance. Assume that this is not the case. Now consider a vertex v and let $N(v)$ denote the neighbors of v in G. Let H denote the complement of the graph induced by the neighbourhood of v, ie the complement of $G[N(v)]$[†] Further, let $C_1, C_2, \ldots, C_r$ denote the components of H. If there are more than k isolated vertices in H, we have a clique of size $\geq k$ in G. Further, since there are at least k anti-edges, using lemma 7, we have a coloring which rainbow connects at least $|E| + k$ pairs of G. Thus we have a *yes* instance.

It remains to deal with the case when the number of isolated vertices in H is less than k. Let C_i be some non-trivial component of H, that is C_i contains at least two vertices. (If no non-trivial component exists, we are already done, since we can bound the number of vertices of G from above by $2k$). We now show a coloring of edges of G such that at least $|C_i| - 1$ non-adjacent vertices are rainbow connected. For this, consider a spanning tree of C_i and color the vertices of the spanning tree level by level using alternate colors. That is, color the root as red, the vertices at the next level in the spanning tree as blue and so on. We map the colors on the vertices of C_i back to the edges of G as follows. If a vertex $u_1 \in C_i$ got the color red, we color the edge $(v, u_1) \in G$ as red. Thus for every edge (u_1, u_2) in C_i that got distinct colors on its end points, we ensure that one pair got rainbow connected via the path u_1, v, u_2. Further, since (u_1, u_2) is an edge in H, it is an anti-edge in G. Thus for every non-trivial component C_i we can rainbow connect $|C_i| - 1$ anti-edges of G. Counting this across all the components we have $\sum_{i=1}^{r} |C_i| - r$ pairs of anti-edges in G rainbow connected. If $\sum_{i=1}^{r} |C_i| - r \geq k$ we have a *yes* instance, otherwise we have:

$$\Sigma_{i=1}^{r} |C_i| - r < k. \tag{1}$$

Let the number of non-trivial components of H be s. Each of these non-trivial components have at least 2 vertices. Hence we have the following:

$$\Sigma_{i=1}^{r} |C_i| \geq 2 * s + (r - s) = r + s \tag{2}$$

Since the number of isolated vertices in H is strictly less than k, we have $r < s + k$. Further, from equations (1) and (2) we get $s < k$. Combining these we have $r < 2k$. Thus we can bound the number of vertices in H as:

$$|H| = \Sigma_{i=1}^{r} |C_i| < r + k < 3k \tag{3}$$

Therefore we have:

$$|G| = |H| + 1 + |O_v| < 3k + 1 + k \implies |G| \leq 4k. \tag{4}$$

Hence, we have a $4k$ vertex kernel. ◀

[†] $G[H]$ denotes the induced subgraph of G on vertices of H

4 Rainbow connectivity on directed graphs

In this section, we consider the rainbow connectivity problem for directed graphs. All the directed graphs considered in this section are assumed to be connected *i.e.*, between any two vertices u, v in the directed graph there is either a directed path from u to v or from v to u. Consider an edge-coloring of a directed graph $G = (V, E)$. We say that there exists a rainbow path between a pair of vertices (u, v) if there exists a directed path from u to v or from v to u with distinct edge colors. An edge coloring of the edges in a directed graph is said to make the graph rainbow connected if between every pair of vertices there is a rainbow path. Analogous to the undirected version, the minimum colors needed to rainbow color a directed graph G is called the rainbow connection number of the directed graph. The rainbow connection number of a directed graph is at least the rainbow connection number of the underlying undirected graph; however, there are examples where the directed graph requires many more colors than the underlying undirected graph. Consider the directed graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $E = \{(v_i, v_{i+1}) : i = 1, \ldots, n-1\} \cup \{(v_1, v_n)\}$. The rainbow connection number of G is $n - 2$ while the rainbow connection number of its underlying undirected graph, which is a cycle, is $\lceil \frac{n}{2} \rceil$.

We study the computational complexity of the problem of computing rainbow connection number for a directed graph. We prove that the problem of deciding whether the rainbow connection of a simple directed graph is at most 2 is NP-hard. As in the case of undirected graphs, we define the problem of subset rainbow connectivity on directed graphs. Given a directed graph $G = (V, E)$ and a set of pairs $P \subseteq V \times V$ decide whether the edges of G can be colored using 2 colors such that every pair in P is rainbow connected (in the directed sense). Throughout this section we will use the term rainbow connected to mean that it is rainbow connected in the directed sense. Our plan, as in the previous cases, is to show that the 2-subset rainbow rainbow connectivity is NP-hard by showing a reduction from the 3SAT problem. We then establish the polynomial time equivalence of 2-subset rainbow connectivity and 2-rainbow connectivity for a directed graph G.

Let $\mathcal{I}$ be an instance of the 3SAT problem with $X = \{x_1, \ldots, x_n\}$ as the set of variables and $C_1, \ldots, C_m$ being the clauses. We construct from $\mathcal{I}$ a directed graph $G = (V, E)$ and a set of pairs $P \subseteq V \times V$ which is an instance of the 2-subset rainbow connectivity problem. For readability sake, we reuse the symbols C_i, x_i to represent the vertices.

$$\begin{aligned} V &= \{C_i : i \in \{1, \ldots, m\}\} \cup X \cup \bar{X} \cup \{T, R, B\} \\ \bar{X} &= \{\bar{x_i} : x_i \in X\} \end{aligned}$$

The edge set E is defined as below. We say that $x_i \in C_j$ to imply that the clause C_j contains the positive occurrence of the variable x_i. If x_i appears negated in the clause C_j we denote it as $\bar{x_i} \in C_j$.

$$\begin{aligned} E = {} & \{(R, T), (T, B)\} \cup \\ & \{(x_i, T), (T, \bar{x_i}), (x_i, \bar{x_i}) : x_i \in X\} \cup \\ & \{(C_j, x_i) : x_i \in C_j\} \cup \\ & \{(\bar{x_i}, C_j) : \bar{x_i} \in C_j\} \end{aligned}$$

The set of pairs P is defined as follows:

$$\begin{aligned} P &= \{(C_i, T) : i \in \{1, \ldots, m\}\} \cup \\ &\quad \{(x_i, C_j), (\bar{x_i}, C_j) : x_i \in C_j\} \cup \\ &\quad \{(x_i, C_j), (\bar{x_i}, C_j) : \bar{x_i} \in C_j\} \cup \\ &\quad \{(R, B)\} \cup \{(R, \bar{x_i}), (B, x_i) : x_i \in X\} \end{aligned}$$

We now state the following lemma (proof in full version [1]) which establishes the correctness of our reduction.

▶ **Lemma 9.** *There exists a satisfying assignment for $\mathcal{I}$ if and only if there is an edge coloring of $G = (V, E)$ with 2 colors such that all the pairs in P are rainbow connected.*

We now prove the equivalence of the following two problems.

▶ **Lemma 10.** *The following two problems are polynomial time equivalent:*
(1) Given a directed graph $G = (V, E)$ decide whether G is 2-rainbow connected.
(2) Given a directed graph $G = (V, E)$, and a set of pairs $P \subseteq V \times V$, decide whether $\langle G, P\rangle$ is 2-subset rainbow connected.

Proof. It suffices to prove that problem (2) reduces to problem (1). Given $\langle G = (V, E), P\rangle$ we construct an instance $G' = (V', E')$ as follows:

$$\begin{aligned} V' &= V \cup V_1 \cup \{v_{ex}\} \\ V_1 &= \{w_{i,j} : (v_i, v_j) \in (V \times V) \setminus P, v_i \neq v_j\} \end{aligned}$$

The edge set E' is defined as:

$$\begin{aligned} E' &= E \cup \{(v_i, w_{i,j}), (w_{i,j}, v_j) : (v_i, v_j) \in (V \times V) \setminus P, v_i \neq v_j\} \cup \\ &\quad \{(v, v_{ex}), (v_{ex}, x) : v \in V, x \in V_1\} \cup E_1 \end{aligned}$$

The set of edges in E_1 are amongst the vertices in V_1 such that the induced subgraph $T = (V_1, E_1)$ is a tournament.

Assume that G has an edge coloring χ using two colors, say *red* and *blue* such that every pair of vertices in P is rainbow connected. We give a coloring χ' the edges of G' as follows:

- Set $\{\chi'(v, v_{ex}) = red : v \in V\}$ and set $\{\chi'(v_{ex}, x) = blue : x \in V_1\}$.
- For every pair $(v_i, v_j) \in (V \times V) \setminus P$, we set $\chi'(v_i, w_{i,j}) = red$ and $\chi'(w_{i,j}, v_j) = blue$.
- Color the edges of the graph induced by V_1 arbitrarily.
- Set $\{\chi'(v_i, v_j) = \chi(v_i, v_j) : v_i \in V, v_j \in V\}$.

It is easy to verify that the above coloring makes G' rainbow connected.

In the other direction, we note that no pair of vertices in P has a directed 2 length path in G' which is not contained entirely in G. Hence if G' has an edge coloring using 2 colors such that every pair has a rainbow path, then the coloring of the induced subgraph G of G' rainbow connects every pair of vertices in P. This completes the proof of the lemma. ◀

Using lemma 9 and lemma 10 we can conclude the following theorem.

▶ **Theorem 11.** *Given a directed graph $G = (V, E)$, it is NP-hard to decide whether G can be colored using two colors such that between every pair of vertices there is a rainbow path.*

5 Conclusion

In this paper, we present several hardness results related to the rainbow connectivity problem. The hardness results for the strong rainbow connectivity and rainbow connectivity problem are due to a series of reductions starting from the vertex coloring problem. Our study on parameterized version of the rainbow connectivity problem shows a linear kernel when we want to maximize the number of pairs which are rainbow connected using two colors. We initiate the study of rainbow connectivity in directed graphs. Further, we show that the problem of deciding whether a directed graph can be rainbow connected using at most 2 colors is NP-hard.

6 Acknowledgements

The first and the second author would like to thank Deepak Rajendraprasad and Dr. L. Sunil Chandran for the useful discussions on the topic.

References

1 P. Ananth, M. Nasre, and K. K. Sarpatwar. Hardness and Parameterized Algorithms on rainbow connectivity problem. *CoRR*, abs/1105.0979, 2011.
2 M. Basavaraju, L.S. Chandran, D. Rajendraprasad, and A. Ramaswamy. Rainbow Connection number and radius. *Arxiv preprint arXiv:1011.0620*, 2010.
3 Y. Caro, A. Lev, Y. Roditty, Z. Tuza, and R. Yuster. On Rainbow Connection. *The Electronic Journal of Combinatorics*, 15(R57):1, 2008.
4 S. Chakraborty, E. Fischer, A. Matsliah, and R. Yuster. Hardness and Algorithms for Rainbow Connection. *Journal of Combinational Optimization*, 21(3):330–347, 2011.
5 L.S. Chandran, A. Das, D. Rajendraprasad, and N.M. Varma. Rainbow connection number and connected dominating sets. *EUROCOMB*, 2011.
6 G. Chartrand, G.L. Johns, K.A. McKeon, and P. Zhang. Rainbow connection in graphs. *Math. Bohem*, 133(1):85–98, 2008.
7 R. G. Downey and M.R. Fellows. *Parameterized complexity*. Monographs in computer science. Springer, 1999.
8 J. Flum and M. Grohe. *Parameterized complexity theory*. Texts in theoretical computer science. Springer, 2006.
9 M. Krivelevich and R. Yuster. The rainbow connection of a graph is (at most) reciprocal to its minimum degree. *J. Graph Theory*, 63:185–191, March 2010.
10 H. Li, X. Li, and S. Liu. The (strong) rainbow connection numbers of cayley graphs of abelian groups. *Arxiv preprint arXiv:1011.0827*, 2010.
11 X. Li and Y. Sun. On strong rainbow connection number. *Arxiv preprint arXiv:1010.6139*, 2010.

Dependence logic with a majority quantifier*

Arnaud Durand[1], Johannes Ebbing[2], Juha Kontinen[3], and Heribert Vollmer[2]

1 **Université Paris Diderot, IMJ, CNRS UMR 7586, Case 7012, 75205 Paris cedex 13, France, durand@logique.jussieu.fr**
2 **Leibniz Universität Hannover, Theoretical Computer Science, Appelstr. 4, 30167 Hannover, Germany, {ebbing,vollmer}@thi.uni-hannover.de**
3 **University of Helsinki, Department of Mathematics and Statistics, P.O. Box 68, 00014, Finland, juha.kontinen@helsinki.fi.**

Abstract

We study the extension of dependence logic $\mathcal{D}$ by a majority quantifier M over finite structures. We show that the resulting logic is equi-expressive with the extension of second-order logic by second-order majority quantifiers of all arities. Our results imply that, from the point of view of descriptive complexity theory, $\mathcal{D}(\mathsf{M})$ captures the complexity class counting hierarchy.

1998 ACM Subject Classification F.1.1 Computability theory, Relations between models; F.1.3 Complexity hierarchies; F.2.2 Computations on discrete structures; F.4.1 Computability theory, Computational logic, Model theory;

Keywords and phrases dependence logic, counting hierarchy, majority quantifier, second order logic, descriptive complexity, finite model theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.252

1 Introduction

We study the extension of dependence logic $\mathcal{D}$ by a majority quantifier M over finite structures. Dependence logic [19] extends first-order logic by dependence atomic formulas

$$=(t_1, \ldots, t_n)$$

the intuitive meaning of which is that the value of the term t_n is completely determined by the values of $t_1, \ldots, t_{n-1}$. While in first-order logic the order of quantifiers solely determines the dependence relations between variables, in dependence logic more general dependencies between variables can be expressed. Historically dependence logic was preceded by partially ordered quantifiers (Henkin quantifiers) of Henkin [8] and Independence-Friendly (IF) logic of Hintikka and Sandu [9]. It is known that both IF logic and dependence logic are equivalent to existential second-order logic ESO in expressive power. From the point of view of descriptive complexity theory, this means that dependence logic captures the class NP.

The framework of dependence logic has turned out be flexible to allow interesting generalizations. For example, the extensions of dependence logic in terms of so-called intuitionistic implication and linear implication was introduced in [1]. In [23] it was shown that extending $\mathcal{D}$ by the intuitionistic implication makes the logic equivalent to full second-order logic SO.

* The third author was supported by grants 127661 and 138163 of the Academy of Finland. The second and fourth author were supported by a grant from DAAD within the PPP programme. The fourth author was also supported by DFG grant VO 630/6-2.

Recently, new variants of the dependence atomic formulas have been introduced in [7] and [6]. Also a modal version of dependence logic was introduced in [20] and has been studied in [14] and [15]. In this paper we are concerned with introducing a new quantifier to dependence logic: the majority quantifier. Adding majority and, more generally, counting capabilities to logical formalisms or computational devices has deserved a lot of attention in theoretical computer science. Understanding the power of counting is an important problem both in logic and in computational complexity:

- The circuit class TC^0, the class of problems solvable by polynomial-size constant-depth circuits with majority gates, is at the current frontier for lower bound techniques (see, e.g., [21]). We have strict separations of classes within TC^0, but above TC^0 we have essentially no lower bounds. By a diagonalization it follows that TC^0 is different from the second level of the exponential-time hierarchy and that uniform TC^0 is strictly included in the class PP of probabilistic polynomial time [2], but a separation from a lower class seems to be far away. In particular, the question if TC^0 equals NC^1 (logarithmic-depth circuits with bounded fan-in gates) is considered the P-NP problem of circuit complexity.
- The counting-hierarchy (the oracle hierarchy built upon PP) can be characterized using majority quantifiers in just the same way as by Wrathall's theorem existential and universal quantifiers characterize the polynomial hierarchy [17].
- By Toda's theorem, one majority quantifier is as powerful as the whole polynomial hierarchy [16].

Here we suggest a definition of a majority quantifier for dependence logic. The proposed semantics mimics that of the existential and universal quantifiers in $\mathcal{D}$. The present paper is devoted to a first study of the resulting logic, denoted by $\mathcal{D}(\mathsf{M})$. We examine some of its basic properties, prove strong normal forms (some of our technically most involved proofs are found here), and show in our main result, that dependence logic with the majority quantifier leads to a new descriptive complexity characterization of the counting hierarchy: $\mathcal{D}(\mathsf{M})$ captures CH.

Engström [5] has also studied generalized quantifiers in dependence logic. He considered different conservative extensions of $\mathcal{D}$—informally this means that he extends $\mathcal{D}$ by generalized quantifiers in a first-order manner. From a descriptive complexity point of view, his logics do not lead out of NP, i.e., ESO, assuming the quantifier in question is ESO-definable (e.g., the majority quantifier). Our approach and results differ from that of Engström since we are in a sense extending dependence logic by a dependence majority quantifier, whose semantics is defined in close analogy with the semantics of $\exists$ and $\forall$ in dependence logic. The results of our paper show that our extension behaves like an extension of SO by second-order generalized quantifiers.

This article is organized as follows. In Sect. 2 we defined dependence logic and discuss some basic results on it. Then we introduce a majority quantifier for the dependence logic setting and discuss the basic properties of $\mathcal{D}(\mathsf{M})$. In Subsect. 2 we discuss the complexity class counting hierarchy and the second-order majority quantifiers Most^k that have been used to characterize it in [10]. In Sect. 3, we introduce second-order majority quantifiers $\mathsf{Most}_{\mathrm{f}}^k$ ranging over functions and in Sect. 4 we show that, for sentences the logics $\mathrm{SO}(\mathsf{Most}_{\mathrm{f}})$ (the extension of second-order logic SO by $\mathsf{Most}_{\mathrm{f}}^k$ for $k \geq 1$) and $\mathcal{D}(\mathsf{M})$ are equivalent. Due to space restrictions, some proofs have to be omitted in this paper, but can be found in the full version at `http://arxiv.org/abs/1109.4750`.

2 Preliminaries

In this section we first define dependence logic and discuss its basic properties. Then we define the counting hierarchy and the logic corresponding to it.

2.1 Dependence Logic

Dependence logic ($\mathcal{D}$) extends the syntax of first-order logic by new dependence atomic formulas. In this article we consider only formulas of $\mathcal{D}$ that are in negation normal form.

▶ **Definition 2.1** ([19]). Let τ be a vocabulary. The τ-formulas of dependence logic ($\mathcal{D}[\tau]$) is defined by extending FO[τ], defined in terms of $\vee$, $\wedge$, $\neg$, $\exists$ and $\forall$, by atomic dependence formulas

$$=(t_1, \ldots, t_n), \tag{1}$$

where $t_1, \ldots, t_n$ are terms.

The meaning of the formula (1) is that the value of the term t_n is functionally determined by the values of the terms $t_1, \ldots, t_{n-1}$. The formula $=()$ is interpreted as $\top$. The semantics of $\mathcal{D}$ will be formally presented shortly.

▶ **Definition 2.2.** Let $\phi \in \mathcal{D}$. The set $\mathrm{Fr}(\phi)$ of free variables of a formula ϕ is defined as for first-order logic, except that we have the new case

$$\mathrm{Fr}(=(t_1, \ldots, t_n)) = \mathrm{Var}(t_1) \cup \cdots \cup \mathrm{Var}(t_n),$$

where $\mathrm{Var}(t_i)$ is the set of variables occurring in term t_i. If $\mathrm{Fr}(\phi) = \emptyset$, we call ϕ a sentence.

The semantics of $\mathcal{D}$ is formulated using the concept of a *Team*. Let $\mathfrak{A}$ be a model with domain A. *Assignments* of $\mathfrak{A}$ are finite mappings from variables into A. The value of a term t in an assignment s is denoted by $t^{\mathfrak{A}}\langle s \rangle$. If s is an assignment, x a variable, and $a \in A$, then $s(a/x)$ denotes the assignment (with domain $\mathrm{dom}(s) \cup \{x\}$) that agrees with s everywhere except that it maps x to a.

▶ **Definition 2.3.** Let A be a set and $\{x_1, \ldots, x_k\}$ a finite (possibly empty) set of variables.

1. A *team* X of A with domain $\mathrm{dom}(X) = \{x_1, \ldots, x_k\}$ (we call A the *co-domain* of X) is any set of assignments $s\colon \{x_1, \ldots, x_k\} \to A$.
2. The relation $\mathrm{rel}(X) \subseteq A^k$ corresponding to X is defined as

$$\mathrm{rel}(X) = \{(s(x_1), \ldots, s(x_k)) : s \in X\}.$$

3. For a function $F\colon X \to A$, we define

$$\begin{aligned} X(F/x) &= \{s(F(s)/x) : s \in X\} \\ X(A/x) &= \{s(a/x) : s \in X \text{ and } a \in A\}. \end{aligned}$$

We will next define the semantics of dependence logic. Below, atomic formulas and their negations are called literals.

▶ **Definition 2.4** ([19]). Let $\mathfrak{A}$ be a model and X a team of A. The satisfaction relation $\mathfrak{A} \models_X \phi$ is defined as follows:

1. If ϕ is a first-order literal, then $\mathfrak{A} \models_X \phi$ iff for all $s \in X$ we have $\mathfrak{A} \models_s \phi$.

2. $\mathfrak{A} \models_X =(t_1, \ldots, t_n)$ iff for all $s, s' \in X$ such that $t_1^{\mathfrak{A}}\langle s\rangle = t_1^{\mathfrak{A}}\langle s'\rangle, \ldots, t_{n-1}^{\mathfrak{A}}\langle s\rangle = t_{n-1}^{\mathfrak{A}}\langle s'\rangle$, we have $t_n^{\mathfrak{A}}\langle s\rangle = t_n^{\mathfrak{A}}\langle s'\rangle$.
3. $\mathfrak{A} \models_X \neg =(t_1, \ldots, t_n)$ iff $X = \emptyset$.
4. $\mathfrak{A} \models_X \psi \wedge \phi$ iff $\mathfrak{A} \models_X \psi$ and $\mathfrak{A} \models_X \phi$.
5. $\mathfrak{A} \models_X \psi \vee \phi$ iff $X = Y \cup Z$ such that $\mathfrak{A} \models_Y \psi$ and $\mathfrak{A} \models_Z \phi$.
6. $\mathfrak{A} \models_X \exists x\psi$ iff $\mathfrak{A} \models_{X(F/x)} \psi$ for some $F\colon X \to A$.
7. $\mathfrak{A} \models_X \forall x\psi$ iff $\mathfrak{A} \models_{X(A/x)} \psi$.

Above, we assume that the domain of X contains the variables free in ϕ. Finally, a sentence ϕ is true in a model $\mathfrak{A}$ (in symbols: $\mathfrak{A} \models \phi$) if $\mathfrak{A} \models_{\{\emptyset\}} \phi$. Above, $A \models_s \phi$ denotes satisfaction in first-order logic.

Let us then recall some basic properties of dependence logic that will be needed later. The following lemma shows that the truth of a $\mathcal{D}$-formula depends only on the interpretations of variables occurring free in the formula. Below, for $V \subseteq \mathrm{dom}(X)$, $X \upharpoonright V$ is defined by

$$X \upharpoonright V := \{s \upharpoonright V \mid s \in X\} \text{ and}$$
$$s \upharpoonright V := \{(a, s(a)) \mid a \in \mathrm{dom}(s) \cap V\}.$$

▶ **Lemma 2.5** ([19]). *Suppose $V \supseteq \mathrm{Fr}(\phi)$. Then $\mathfrak{A} \models_X \phi$ if and only if $\mathfrak{A} \models_{X\upharpoonright V} \phi$.*

All formulas of dependence logic also satisfy the following strong monotonicity property called *Downward Closure*.

▶ **Proposition 2.6** ([19]). *Let ϕ be a formula of dependence logic, $\mathfrak{A}$ a model, and $Y \subseteq X$ teams. Then $\mathfrak{A} \models_X \phi$ implies $\mathfrak{A} \models_Y \phi$.*

On the other hand, the expressive power of sentences of $\mathcal{D}$ coincides with that of existential second-order sentences:

▶ **Theorem 2.7** ([19]). $\mathcal{D} = \mathrm{ESO}$.

Finally, we note that dependence logic is a conservative extension of first-order logic.

▶ **Definition 2.8.** A formula ϕ of $\mathcal{D}$ is called a first-order formula if it does not contain dependence atomic formulas as subformulas.

First-order formulas of dependence logic satisfy the so-called *flatness* property:

▶ **Theorem 2.9** ([19]). *Let ϕ be a first-order formula of dependence logic. Then for all $\mathfrak{A}$ and X:*

$$\mathfrak{A} \models_X \phi \text{ if and only if for all } s \in X \text{ we have } \mathfrak{A} \models_s \phi.$$

2.2 Dependence logic with a majority quantifier

The main topic of the present paper is the study of a logic obtained from $\mathcal{D}$ by the introduction of a majority quantifier M. We denote this extended logic by $\mathcal{D}(\mathsf{M})$. It is formally defined by extending the syntax and semantics of dependence logic by the following clause:

$\mathfrak{A} \models_X \mathsf{M}x\phi(x)$ iff for at least $|A|^{|X|}/2$ many functions $F\colon X \to A$ we have $\mathfrak{A} \models_{X(F/x)} \phi(x)$.

Analogously to $\mathcal{D}$ the logic $\mathcal{D}(\mathsf{M})$ has the so-called empty team property:

▶ **Proposition 2.10.** *For all models $\mathfrak{A}$ and formulas ϕ of $\mathcal{D}(\mathsf{M})$, it holds that $\mathfrak{A} \models_\emptyset \phi$.*

Proof. The claim is proved using induction on ϕ. ◀

We also observe that $\mathcal{D}(\mathsf{M})$ satisfies the downward closure property (compare to Proposition 2.6).

▶ **Proposition 2.11.** *Let ϕ be a formula of $\mathcal{D}(\mathsf{M})$, $\mathfrak{A}$ a model, and $Y \subseteq X$ teams. Then $\mathfrak{A} \models_X \phi$ implies $\mathfrak{A} \models_Y \phi$.*

Proof. The claim is proved using induction on ϕ. We consider the case where ϕ is $\mathsf{M}x\psi$. The other cases are proved exactly as for dependence logic (see Proposition 3.10 in [19]). By the induction assumption, ψ satisfies the claim. Let $\mathfrak{A}$, X and Y be as above and suppose that $|A| = n$, $|X| = m$, and $|Y| = m - 1$. Let us assume $\mathfrak{A} \models_X \phi$. Then for at least $(n^m)/2$ many functions $F\colon X \to A$ it holds that $\mathfrak{A} \models_{X(F/x)} \psi$. Since ψ satisfies the claim, it holds that if $\mathfrak{A} \models_{X(F/x)} \psi$, then $\mathfrak{A} \models_{Y(F'/x)} \psi$, where

$$F' = F \upharpoonright Y. \tag{2}$$

Note that, in the worst case, at most n different functions F gives rise to the same reduct F' in (2). Therefore, the number of functions $F\colon Y \to A$ satisfying $\mathfrak{A} \models_{Y(F/x)} \psi$ is at least $(n^m)/2n = n^{m-1}/2$ and hence $\mathfrak{A} \models_Y \phi$. It is easy to see that the analogous argument can be used with any $Y \subseteq X$. ◀

A well-studied property in the context of dependence logic is that of *coherence*, defined as follows. A formula ϕ is called *k-coherent* if and only if for all structures $\mathfrak{A}$ and teams X it holds that

$$\mathfrak{A} \models_X \phi \Leftrightarrow \text{ for every } k\text{-element subteam } X' \subseteq X \text{ it holds that } \mathfrak{A} \models_{X'} \phi.$$

1-coherent formulas are also called *flat*.

▶ **Proposition 2.12.** *There is a formula $\phi \in \mathcal{D}(\mathsf{M})$ without dependence atoms such that ϕ is not k-coherent for any $k \in \mathbb{N}$.*

We also note that the analogue of Proposition 2.5 does not hold for $\mathcal{D}(\mathsf{M})$.

▶ **Proposition 2.13.** *The truth of a $\mathcal{D}(\mathsf{M})$-formula ϕ may depend on the interpretations of variables that do not occur free in ϕ.*

2.3 Second-order Majority Quantifiers and the Counting Hierarchy

In this section we define the counting hierarchy and the relevant generalized quantifiers.

▶ **Definition 2.14.** Let $k \geq 1$. We define the k-ary second-order generalized quantifier Most^k binding a k-ary relation symbol X in a formula ϕ. Assume $\mathfrak{A}$ is a structure with domain A such that $|A| = n$. Then the semantics of this quantifier is defined as follows:

$$\mathfrak{A} \models \mathsf{Most}^k X\phi(X) \iff \big|\{B \subseteq A^k \mid \mathfrak{A} \models \phi(B)\}\big| \geq 2^{n^k}/2.$$

We will also make use of the so-called k-ary second-order *Rescher quantifier*, defined as follows:

$$\mathfrak{A} \models \mathsf{R}^k X, Y(\phi(X), \psi(Y)) \iff \big|\{B \subseteq A^k \mid \mathfrak{A} \models \phi(B)\}\big| \geq \big|\{B \subseteq A^k \mid \mathfrak{A} \models \psi(B)\}\big|.$$

It is quite easy to see that the Most^k-quantifier can be defined in terms of the quantifier R^k. In [10] it was shown that the k-ary Rescher quantifier R^k can be defined in first order logic with Most^{k+1}, and, for $k \geq 2$, already with Most^k. It is worth noting that in [10] the quantifiers Most^k and R^k are interpreted as strict majority and strict inequality, respectively. All the results of [10] that we use also hold under the "non-strict" interpretation adopted in this article.

The counting hierarchy (CH) is the analogue of the polynomial hierarchy, defined as the oracle hierarchy using as building block probabilistic polynomial time (the class PP) instead of NP:

1. $\mathrm{C}_0\mathrm{P} = \mathrm{P}$,
2. $\mathrm{C}_{k+1}\mathrm{P} = \mathrm{PP}^{\mathrm{C}_k\mathrm{P}}$,
3. $\mathrm{CH} = \bigcup_{k\in\mathbb{N}} \mathrm{C}_k\mathrm{P}$.

The counting hierarchy was first defined by Wagner [22] but the above equivalent formulation is due to Torán [17]. The class PP consists of languages L for which there is a polynomial time-bounded nondeterministic Turing machine N such that, for all inputs x, $x \in L$ iff more than half of the computations of N on input x accept.

In [10] it was shown that the extension FO(Most) of FO by the quantifiers Most^k, for $k \in \mathbb{N}$, describes exactly the problems in the counting hierarchy. The proof therein used the fact that the second-order existential quantifier can be simulated by Most^k and first-order logic.

▶ **Theorem 2.15.** $\mathrm{FO}(\mathsf{Most}) = \mathrm{SO}(\mathsf{Most}) = \mathrm{CH}$.

By the above remark we see that in the previous theorem the Most quantifiers can be replaced by Rescher quantifiers.

3 Majority over Functions

For our main result that compares second-order logic and dependence logic with majority-quantifiers, it turns out to be helpful to consider a version of the Most-quantifier that ranges over functions instead of relations.

▶ **Definition 3.1.** Let $k \geq 1$. We define the k-ary second-order generalized quantifier $\mathsf{Most}_\mathrm{f}^k$ binding a k-ary function symbol g in a formula ϕ. Assume $\mathfrak{A}$ is a structure with domain A such that $|A| = n$. Then

$$\mathfrak{A} \models \mathsf{Most}_\mathrm{f}^k g\ \phi(g) \iff \left|\{f\colon A^k \to A \mid \mathfrak{A} \models \phi(f)\}\right| \geq n^{n^k}/2.$$

We denote by $\mathrm{SO}(\mathsf{Most}_\mathrm{f})$ the extension of SO by the quantifiers $\mathsf{Most}_\mathrm{f}^k$ for all $k \geq 1$. The following elementary properties of $\mathrm{SO}(\mathsf{Most}_\mathrm{f})$ will be useful.

▶ **Proposition 3.2.** *The following equivalences hold:*

1. $(\phi \vee \mathsf{Most}_\mathrm{f}^k g\,\psi) \equiv \mathsf{Most}_\mathrm{f}^k g\,(\phi \vee \psi)$, *if g does not appear free in ϕ,*
2. $(\phi \wedge \mathsf{Most}_\mathrm{f}^k g\,\psi) \equiv \mathsf{Most}_\mathrm{f}^k g\,(\phi \wedge \psi)$, *if g does not appear free in ϕ.*

Note that since also $\neg\mathsf{Most}_\mathrm{f}^k g\,\psi$ is equivalent to $\mathsf{Most}_\mathrm{f}^k g\,\neg\psi$, Proposition 3.2 allows us to transform formulas of $\mathrm{SO}(\mathsf{Most}_\mathrm{f})$ to prenex normal form. The equivalences of Proposition 3.2 obviously hold also for the relational majority quantifiers Most^k.

The next proposition states the intuitively obvious fact that the extensions of SO by the quantifiers Most^k or alternatively by $\mathsf{Most}_\mathrm{f}^k$, for $k \in \mathbb{N}$, are equal in expressive power.

▶ **Proposition 3.3.** $\mathrm{SO}(\mathsf{Most}) = \mathrm{SO}(\mathsf{Most}_\mathrm{f})$.

Proof. We prove the claim by an argument analogous to Theorem 3.4 in [10]. We will show how to express the quantifier $\mathsf{Most}_\mathrm{f}^k$ in the logic $\mathrm{SO}(\mathsf{Most})$ implying $\mathrm{SO}(\mathsf{Most}_\mathrm{f}) \leq \mathrm{SO}(\mathsf{Most})$. The converse inclusion is proved analogously.

Let us consider a formula of the form $\mathsf{Most}_\mathrm{f}^k g\phi(g) \in \mathrm{SO}(\mathsf{Most}_\mathrm{f})$. Let $\mathfrak{A}$ be a structure. We may assume that $\mathfrak{A}$ is ordered (we can existentially quantify it) and hence there is a FO-formula $\delta(\overline{x},\overline{y})$ defining the lexicographic ordering of the set A^{k+1}. We can construct a formula $\chi(X,Y)$ which, for $A_1, A_2 \subseteq A^{k+1}$, defines the lexicographic ordering ($A_1 \leq_l A_2$) of $k+1$-ary relations induced by $\delta(\overline{x},\overline{y})$.

It is now fairly straightforward to express $\mathsf{Most}_\mathrm{f}^k g\phi(g)$ in the logic $\mathrm{SO}(\mathsf{Most})$. Let

$$\begin{aligned} G &= \{B \subseteq A^{k+1} \mid B \text{ is the graph of some } g \text{ and } \mathfrak{A} \models \phi(g)\},\\ G^c &= \{B \subseteq A^{k+1} \mid B \text{ is the graph of some } g \text{ and } \mathfrak{A} \not\models \phi(g)\}. \end{aligned}$$

It now suffices to express $|G| \geq |G^c|$ in the logic $\mathrm{SO}(\mathsf{Most})$. For a $D \subseteq A^{k+1}$, define the set $\mathrm{IS}(D)$ (the "initial segment" determined by D) by

$$\mathrm{IS}(D) = \{D' \subseteq A^{k+1} \mid D' \notin G \cup G^c \text{ and } D' \leq_l D\}.$$

The condition $|G| \geq |G^c|$ can be now expressed by

$$\forall D\big(|G^c \cup \mathrm{IS}(D)| \geq 2^{n^{k+1}}/2 \Rightarrow |G \cup \mathrm{IS}(D)| \geq 2^{n^{k+1}}/2\big).$$

It is straightforward to express this in the logic $\mathrm{SO}(\mathsf{Most})$. ◀

The following lemma will be needed in the proof of the next proposition.

▶ **Lemma 3.4.** *Let $k \geq 1$. There exists an* ESO *sentence $\chi(g)$, where g is k-ary, such that for all $\mathfrak{A}$ with domain $|A| = n$, $\chi(g)$ is satisfied by exactly $\lceil n^{n^k}/2 \rceil - 2^{n^k-1}$ many k-ary functions g none of which is a characteristic function of some k-ary relation.*

The next proposition gives a useful normal form for sentences of the logic $\mathrm{SO}(\mathsf{Most}_\mathrm{f})$.

▶ **Proposition 3.5.** *Every sentence of* $\mathrm{SO}(\mathsf{Most}_\mathrm{f})$ *is equivalent to a sentence of the form*

$$\exists \overline{h}^1 \mathsf{Most}_\mathrm{f}^k g_1 \cdots \mathsf{Most}_\mathrm{f}^k g_l \, \exists \overline{h}^2 \theta,$$

where the function symbols in $\overline{h}^1$, and g_i for $1 \leq i \leq l$, are k-ary ($k \geq 3$), and θ is a universal first-order sentence.

Proof. Note that by Proposition 3.3 it suffices to show that every sentence of the logic $\mathrm{SO}(\mathsf{Most})$ can be transformed to this form. The result in [10] shows (as pointed out in Lemma 10.5 in [11]) that, in the presence of built-in relations $\{<,+,\times\}$, sentences of $\mathrm{SO}(\mathsf{Most})$ can be assumed to have the form

$$\mathsf{Most}^{i_1} Y_1 \cdots \mathsf{Most}^{i_l} Y_l \, \psi, \tag{3}$$

where ψ is first-order. Furthermore, when l in (3) is fixed, we get a fragment of $\mathrm{SO}(\mathsf{Most})$ characterizing the lth level of CH, i. e., the class $\mathrm{C}_l\mathrm{P}$.

We will next show how to transform any sentence of the form (3) to the required form. The first step is to quantify out the built-in relations $\{<,+,\times\}$ to get a sentence of the form

$$\exists X_< \exists X_+ \exists X_\times \mathsf{Most}^{i_1} Y_1 \cdots \mathsf{Most}^{i_l} Y_l \, \psi^*. \tag{4}$$

The relations $X_<$, X_+, and $X_\times$ can be axiomatized as part of ψ^* (compare to case 2 of Proposition 3.2). Then we modify the sentence (4) to change the arities of all the quantified relations to some big enough k. We need only to replace all occurrences, say $Y_i(t_1,\dots,t_{i_j})$, of the quantified relation symbols in ψ^* by $Y_i(t_1,\dots,t_{i_j},0,\dots,0)$. (Note that the needed constant 0 can be defined using the linear order.) Increasing the arity of the second-order existential quantifiers in (4) is clearly unproblematic. For the majority quantifiers Most^{i_j}, we note that for any structure $\mathfrak{A}$ of cardinality n and $B \subseteq A^v$, the number of k-ary relations $D \subseteq A^k$ such that

$$\{\overline{a} \in A^v \mid (\overline{a},0,\dots,0) \in D\} = B \tag{5}$$

is $2^{n^k-n^v}$, which is independent of B. Furthermore, obviously the truth of ψ^* with respect to a tuple of k-ary relations $D_1,\dots,D_{l+3}$ only depends on whether $\psi^*(B_1,\dots,B_{l+3})$ holds, where B_i is the restriction of D_i defined analogously to (5). This fact allows us to increase also the arity of the majority quantifiers without changing the meaning of the sentence (4).

Let us then show how to transform the relational quantifiers in (4) into function quantifiers. We claim that it is possible to replace $\psi^*(X_<,X_+,X_\times,Y_1,\dots,Y_l)$ by a formula of the form

$$\theta(\overline{g}) \vee (\forall\overline{x}(\bigwedge_{1\le i\le l} g_i(\overline{x}) \in \{0,1\}) \wedge \psi'(g_</X_<, g_+/X_+, g_\times/X_\times, g_1/Y_1,\dots,g_l/Y_l)), \tag{6}$$

where $\overline{g} = (g_<,g_+,g_\times,g_1,\dots,g_l)$, the new function symbols are all k-ary and ψ' is obtained from ψ^* by substituting subformulas $Z(t_1,\dots,t_k)$ by the corresponding $g_{(.)}(1_1,\dots,t_k) = 1$, where $Z \in \{Y_1,\dots,Y_l,X_<,X_+,X_\times\}$.

The formula $\theta(\overline{g})$ is a ESO-formula that accepts certain dummy functions in order to shift the border of acceptance from $(2^{|\mathfrak{A}|^k})/2$ (half of k-ary relations) to $|\mathfrak{A}|^{|\mathfrak{A}|^k}/2$ (half of k-ary functions). The logical form of θ is

$$\chi(g_1) \vee \chi(g_2) \vee \cdots \vee \chi(g_l),$$

where $\chi(g)$ is defined in Lemma 3.4. Note that we repeatedly use case 1 of Lemma 3.2 to gather all the formulas $\chi(g_i)$ into θ which is placed after the block of all majority quantifiers.

To prove the claim we finally transform the formula (6) into Skolem normal form to get a sentence of the form

$$\exists g_<\exists g_+\exists g_\times \mathsf{Most}^k_{\mathrm{f}} g_1 \cdots \mathsf{Most}^k_{\mathrm{f}} g_l\, \exists\overline{g}\psi', \tag{7}$$

where ψ' is a universal FO-sentence. ◀

4 $\mathrm{SO}(\mathsf{Most}) = \mathcal{D}(\mathsf{M})$

In this section we show that the logics $\mathrm{SO}(\mathsf{Most}_{\mathrm{f}})$ (and thus, by the previous section, $\mathrm{SO}(\mathsf{Most})$) and $\mathcal{D}(\mathsf{M})$ are equivalent with respect to sentences.

We will first show a compositional translation mapping formulas of $\mathcal{D}(\mathsf{M})$ into sentences of $\mathrm{SO}(\mathsf{Most}_{\mathrm{f}})$. This translation is analogous to the translation from $\mathcal{D}$ into ESO of [19].

▶ **Lemma 4.1.** *Let τ be a vocabulary. For every $\mathcal{D}(\mathsf{M})[\tau]$-formula ϕ there is a $\tau \cup \{S\}$-sentence ψ of $\mathrm{SO}(\mathsf{Most}_{\mathrm{f}})$ such that for all models $\mathfrak{A}$ and teams X with $\mathrm{dom}(X) = \mathrm{Fr}(\phi)$ it holds that*

$$\mathfrak{A} \models_X \phi \iff (\mathfrak{A}, \mathrm{rel}(X)) \models \psi.$$

Proof. For technical reasons to be motived shortly, we will actually prove a slightly more general result showing that for every $\mathcal{D}(\mathsf{M})[\tau]$-formula ϕ and every finite set of variables $\{y_1, \ldots, y_n\} \supseteq \mathrm{Fr}(\phi)$ there is a $\mathrm{SO}(\mathsf{Most}_\mathrm{f})[\tau \cup S]$-sentence ψ such that for all $\mathfrak{A}$ and teams X with $\mathrm{dom}(X) = \{y_1, \ldots, y_n\}$ it holds that

$$\mathfrak{A} \models_X \phi \iff (\mathfrak{A}, \mathrm{rel}(X)) \models \psi.$$

We will prove the claim using induction on the structure of $\mathcal{D}(\mathsf{M})$-formulas. In the following we write $\phi(y_1, \ldots, y_n)$ to mean that $\mathrm{Fr}(\phi) \subseteq \{y_1, \ldots, y_n\}$. The quantifiers R^k can be uniformly defined in the logic $\mathrm{SO}(\mathsf{Most})$, hence by the results of the previous section, also in $\mathrm{SO}(\mathsf{Most}_\mathrm{f})$. Therefore, we may freely use the quantifiers R^k in the translation.

Atomic formulas and their negations are translated exactly in the same way as in the analogous translation from $\mathcal{D}$ into ESO in [19]. The cases $\gamma := \exists y_n \phi(y_1, \ldots, y_n)$ and $\gamma := \forall y_n \phi(y_1, \ldots, y_n)$ are also translated as in [19]. Suppose then that $\gamma := (\phi \vee \psi)(y_1, \ldots, y_n)$ and that $\phi^*(S)$ and $\psi^*(S)$ already exist by induction hypothesis. We translate γ as follows:

$$\gamma^*(S) := \exists Y \exists Z(\phi^*(Y/S) \wedge \psi^*(Z/S) \wedge \forall y_1 \ldots \forall y_n(S(\overline{y}) \to R(\overline{y}) \vee T(\overline{y}))). \tag{8}$$

Note that $\gamma^*(S)$ is defined as in [19]. The only difference is that in the case of dependence logic the sentence (8) can be written using a single sentence $\phi^*(S)$ (and $\psi^*(S)$) that translates ϕ over teams with domain $\mathrm{Fr}(\phi)$ (see Proposition 2.5). In the case of $\mathcal{D}(\mathsf{M})$ the behavior of ϕ and ψ over teams X with $\mathrm{dom}(X) = \{y_1, \ldots, y_n\}$ does not in general reduce to their behavior over $X \upharpoonright \mathrm{Fr}(\phi)$ and $X \upharpoonright \mathrm{Fr}(\psi)$ (see Proposition 2.13). Therefore, to formulate the sentence (8), we need sentences $\phi^*(S)$ and $\psi^*(S)$ that are correct translations of ϕ and ψ with respect to teams with domain $\{y_1, \ldots, y_n\}$.

The case $\gamma := (\phi \wedge \psi)(y_1, \ldots, y_n)$ is also analogous to [19]. It remains to consider the case where our formula γ is of the form

$$\gamma := \mathsf{M} y_n \phi(y_1, \ldots, y_n) \tag{9}$$

and ϕ is a formula for which we have already a translation into an $\mathrm{SO}(\mathsf{Most}_\mathrm{f})[\tau \cup S]$ sentence $\phi^*(S)$. We claim that γ can be translated as follows:

$$\gamma^*(S) := \mathsf{R}^n Y, Z(\theta_1(Y), \theta_2(Z)) \tag{10}$$

where

$$\theta_1(Y) := \phi^*(Y/S) \wedge \forall y_1 \ldots \forall y_{n-1} \exists^{=1} y_n Y(\overline{y}) \wedge \forall y_1 \ldots \forall y_{n-1}(\exists y_n Y(\overline{y}) \leftrightarrow S(\overline{y}'))$$
$$\theta_2(Z) := \neg\phi^*(Z/S) \wedge \forall y_1 \ldots \forall y_{n-1} \exists^{=1} y_n Z(\overline{y}) \wedge \forall y_1 \ldots \forall y_{n-1}(\exists y_n Z(\overline{y}) \leftrightarrow S(\overline{y}'))$$
$$\text{and } \overline{y}' := y_1, \ldots, y_{n-1}$$

The following equivalence is now obvious for all $\mathfrak{A}$ and X:

$$\mathfrak{A} \models_X \gamma \Leftrightarrow (\mathfrak{A}, \mathrm{rel}(X)) \models \gamma^*(S).$$

◀

Next we will show that, for sentences, Lemma 4.1 can be reversed.

▶ **Lemma 4.2.** *Let τ be a vocabulary and $\phi \in \mathrm{SO}(\mathsf{Most}_\mathrm{f})[\tau]$. Then there is a sentence $\psi \in \mathcal{D}(\mathsf{M})[\tau]$ such that for all models $\mathfrak{A}$:*

$$\mathfrak{A} \models \phi \iff \mathfrak{A} \models \psi.$$

Proof. By Proposition 3.5 we may assume that ϕ is of the form:

$$\exists \overline{h}^1 \mathsf{Most}_{\mathrm{f}}^k g_1 \cdots \mathsf{Most}_{\mathrm{f}}^k g_n \, \exists \overline{h}^2 \forall x_1 \cdots \forall x_m \psi, \tag{11}$$

where the function symbols in $\overline{h}^1$ and $g_1, \ldots, g_n$ are k-ary, and ψ is quantifier free. Before translating this sentence into $\mathcal{D}(\mathsf{M})$, we will first apply certain reductions to it. First of all, we make sure that the functions g_i have only occurrences of the form $g_i(\overline{x})$ in ψ where $\overline{x} = (x_1, \ldots, x_k)$. We can achieve this by existentially quantifying new names f_i for these symbols and passing on to the sentence

$$\exists \overline{h}^1 \mathsf{Most}_{\mathrm{f}}^k g_1 \cdots \mathsf{Most}_{\mathrm{f}}^k g_n \, \exists \overline{h}^2 \exists f_1 \cdots \exists f_n \forall x_1 \cdots \forall x_m (\bigwedge_{1 \le j \le n} g_j(\overline{x}) = f_j(\overline{x}) \wedge \psi^*), \tag{12}$$

where ψ^* is obtained from ψ by replacing all occurrences of g_i by f_i for $1 \le j \le n$. Analogously, we may also assume that the functions h in $\overline{h}^1$ have only occurrences $h(x_1, \ldots, x_k)$ in ψ. Here m can always be made at least k.

The next step is to transform the quantifier-free part ψ^* to satisfy the condition that for each of the function symbols h in $\overline{h}^2$ (also f_i) there is a unique tuple $\overline{x}$ of pairwise distinct variables such that all occurrences of it in ψ^* are of the form $h(\overline{x})$ ($f_i(\overline{x})$). In order to achieve this, we might have to introduce new existentially quantified functions and also universal first-order quantifiers (see Theorem 6.15 in [19]), but the quantifier structure of the sentence (11) does not change.

We will now assume that the sentence (11) has the properties discussed above:

1. The function symbols $h \in \overline{h}^1$ and g_i have only occurrences of the form $h(x_1, \ldots, x_k)$ and $g_i(x_1, \ldots, x_k)$ in ψ, respectively.
2. For each h in $\overline{h}^2$ (f_i, for $1 \le i \le n$) there is a unique tuple $\overline{x}$ of pairwise distinct variables such that all occurrences of h in ψ^* are of the form $h(\overline{x})$ ($f_i(\overline{x})$).

We will now show how the sentence (11) can be translated into $\mathcal{D}(\mathsf{M})$. For the sake of bookkeeping, we assume that $\overline{h}^1 = h_1 \ldots h_p$, $\overline{h}^2 = h_{p+1} \ldots h_r$, and that h_i appears in ψ only as $h_i(\overline{x}^i)$. We claim now that the following sentence of $\mathcal{D}(\mathsf{M})$ is a correct translation for (11):

$$\forall x_1 \cdots \forall x_k \exists y_1 \cdots \exists y_p \mathsf{M} z_1 \cdots \mathsf{M} z_n \forall x_{k+1} \cdots \forall x_m \exists y_{p+1} \cdots \exists y_r (\bigwedge_{p+1 \le j \le r} {=}(\overline{x}^i, y_i) \wedge \theta), \tag{13}$$

where θ is obtained from ψ by replacing all occurrences of the term $g_i(x_1, \ldots, x_k)$ by the variable z_i and, similarly, each occurrence of $h_i(\overline{x}^i)$ by y_i.

Let us then show that the sentence ϕ (see (11)) and sentence (13) are logically equivalent. Let $\mathfrak{A}$ be a structure and let $\mathbf{h}_1, \ldots, \mathbf{h}_r$ and $\mathbf{g}_1, \ldots, \mathbf{g}_n$ interpret the corresponding function symbols. We will show that the following holds:

$$(\mathfrak{A}, \overline{\mathbf{h}}, \overline{\mathbf{g}}) \models_X \psi \Leftrightarrow \mathfrak{A} \models_{X^*} \theta, \tag{14}$$

where $X = \{\emptyset\}(A/x_1) \cdots (A/x_m)$ and

$$\begin{array}{rccccccc} X^* = & \{\emptyset\}(A/x_1) & \cdots & (A/x_k) & (H_1/y_1) & \cdots & (H_p/y_p) \\ & (G_1/z_1) & \cdots & (G_n/z_n) & (A/x_{k+1}) & \cdots & (A/x_m) \\ & (H_{p+1}/y_1) & \cdots & (H_r/y_r), & & & \end{array}$$

where the supplement functions H_i and G_i are defined using the functions $\mathbf{h}_i$ and $\mathbf{g}_i$ as follows:

$$\begin{aligned} H_i(s) &= \mathbf{h}_i(s(x_1),\dots,s(x_k)) \text{ for } 1 \le i \le p, \\ H_i(s) &= \mathbf{h}_i(s(\overline{x}^i)) \text{ for } p+1 \le i \le r, \\ G_i(s) &= \mathbf{g}_i(s(x_1),\dots,s(x_k)) \text{ for } 1 \le i \le n, \end{aligned}$$

and where $s(\overline{x}^i)$ is the tuple obtained by pointwise application of s. The claim in (14) is now proved using induction on the structure of the quantifier-free formula ψ. Note that ψ is a first-order formula of dependence logic; hence, by Theorem 2.9, (14) holds iff the equivalence holds for each $s \in X$ (equivalently $s \in X^*$ since the values of the universally quantified variables functionally determine the values of all the other variables) individually. We can now show, using induction on the construction of ψ, that for all $s \in X^*$ it holds that

$$\mathfrak{A} \models_s \theta \iff (\mathfrak{A}, \overline{\mathbf{h}}, \overline{\mathbf{g}}) \models_{s'} \psi, \tag{15}$$

where $s' = s \upharpoonright \{x_1,\dots,x_m\}$. The key to this result is the fact that, for every s, the interpretation of the variables z_i and y_i agree with the interpretation of the terms $h_i(\overline{x}^i)$ and $g(x_1,\dots,x_k)$, respectively.

Finally, we note that there is a one-to-one correspondence between all possible interpretations $\mathbf{h}_1,\dots,\mathbf{h}_r$ and $\mathbf{g}_1,\dots,\mathbf{g}_n$ for the function symbols and teams X^* satisfying the dependence atomic formulas in (13). Therefore, sentence ϕ (see (11)) and sentence (13) are logically equivalent. ◀

5 Conclusion and Open Questions

We have seen that extending dependence logic by a majority quantifier increases the expressive power of dependence logic considerably. One particular consequence of our result is that $\mathcal{D}(\mathsf{M})$ is closed under classical negation on the level of sentences. Note further that, for open formulas, this does not hold because of the downward closure property of formulas.

Several open questions remain and we now discuss some of them. Firstly, Proposition 2.12 shows that the fragment of $\mathcal{D}(\mathsf{M})$ without dependence atoms does not satisfy the flatness property. It would be interesting to pin down the exact expressive power of sentences of $\mathcal{D}(\mathsf{M})$ without dependence atoms.

The second open question concerns the open formulas of $\mathcal{D}(\mathsf{M})$. In [12] it was shown that the open formulas of $\mathcal{D}$ correspond to the downwards monotone properties of NP (see [12] for the exact formulation). We conjecture that the open formulas of $\mathcal{D}(\mathsf{M})$ correspond in an analogous manner to the downwards monotone properties of CH.

The majority quantifier is only one particular example of so-called *generalized quantifiers* (or, *Lindström quantifiers*), introduced in [13] and studied extensively in the context of descriptive complexity theory (see surveys [18] and [4]). In [3], second-order Lindström quantifiers were introduced and some results concerning their expressive power were obtained. We consider it an interesting study to enrich in a similar way dependence logic by further generalized quantifiers and relate the obtained logics to those studied in [3].

References

1 S. Abramsky and J. Väänänen. From IF to BI. *Synthese*, 167(2):207–230, 2009.

2 E. Allender. The permanent requires large uniform threshold circuits. *Chicago J. Theoret. Comput. Sci.*, pages Article 7, 19 pp. (electronic), 1999.

3 H.-J. Burtschick and H. Vollmer. Lindström quantifiers and leaf language definability. *Int. J. Found. Comput. Sci.*, 9(3):277–294, 1998.

4 H.-D. Ebbinghaus and J. Flum. *Finite model theory, 2nd edition*. Perspectives in Mathematical Logic. Springer-Verlag, 1999.

5 F. Engström. Generalized quantifiers in dependence logic. arXiv:1103.0396.

6 P. Galliani. Inclusion and exclusion dependencies in team semantics: On some logics of imperfect information. arXiv:1106.1323.

7 E. Grädel and J. Väänänen. Dependence and independence. To appear in *Studia Logica*.

8 L. Henkin. Some remarks on infinitely long formulas. In *Infinitistic Methods (Proc. Sympos. Foundations of Math., Warsaw, 1959)*, pages 167–183. Pergamon, Oxford, 1961.

9 J. Hintikka and G. Sandu. Informational independence as a semantical phenomenon. In *Logic, methodology and philosophy of science, VIII (Moscow, 1987)*, volume 126 of *Stud. Logic Found. Math.*, pages 571–589. North-Holland, Amsterdam, 1989.

10 J. Kontinen. A logical characterization of the counting hierarchy. *ACM Trans. Comput. Log.*, 10(1), 2009.

11 J. Kontinen and H. Niemistö. Extensions of MSO and the monadic counting hierarchy. *Inf. Comput.*, 209(1):1–19, 2011.

12 J. Kontinen and J. Väänänen. On definability in dependence logic. *J. Log. Lang. Inf.*, 18(3):317–332, 2009.

13 P. Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

14 P. Lohmann and H. Vollmer. Complexity results for modal dependence logic. In A. Dawar and H. Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech*, volume 6247 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2010.

15 M. Sevenster. Model-theoretic and computational properties of modal dependence logic. *J. Log. Comput.*, 19(6):1157–1173, 2009.

16 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

17 J. Torán. Complexity classes defined by counting quantifiers. *J. Assoc. Comput. Mach.*, 38(3):753–774, 1991.

18 J. Väänänen. Generalized quantifiers, an introduction. In *Generalized quantifiers and computation (Aix-en-Provence, 1997)*, volume 1754 of *Lecture Notes in Comput. Sci.*, pages 1–17. Springer, Berlin, 1999.

19 J. Väänänen. *Dependence logic: A New Approach to Independence Friendly Logic*, volume 70 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 2007.

20 J. Väänänen. Modal dependence logic. In K. Apt and R. van Rooij, editors, *New Perspectives on Games and Interaction*, volume 5 of *Texts in Logic and Games*, pages 237–254. Amsterdam University Press, 2008.

21 H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.

22 K. Wagner. The complexity of combinatorial problems with succint input representation. *Acta Informatica*, 23:325–356, 1986.

23 F. Yang. Expressing second-order sentences in intuitionistic dependence logic. To appear in *Studia Logica*.

Modal Logics Definable by Universal Three-Variable Formulas

Emanuel Kieroński*, **Jakub Michaliszyn***, **and Jan Otop**

Institute of Computer Science
University of Wrocław
{kiero,jmi,jotop}@cs.uni.wroc.pl

Abstract

We consider the satisfiability problem for modal logic over classes of structures definable by universal first-order formulas with three variables. We exhibit a simple formula for which the problem is undecidable. This improves an earlier result in which nine variables were used. We also show that for classes defined by three-variable, universal Horn formulas the problem is decidable. This subsumes decidability results for many natural modal logics, including T, B, $\mathsf{K4}$, $\mathsf{S4}$, $\mathsf{S5}$.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases modal logic, decidability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.264

1 Introduction

Modal logic for almost a hundred year has been an important topic in many academic disciplines, including philosophy, mathematics, linguistics, and computer science. Currently it seems to be most intensively investigated by computer scientists. Among numerous branches in which modal logic, sometimes in disguise, finds applications, are hardware and software verification, cryptography and knowledge representation.

Modal logic was introduced by philosophers to study modes of truth. The idea was to extend propositional logic by some new constructions, of which two most important were $\Diamond\varphi$ and $\Box\varphi$, originally read as φ *is possible* and φ *is necessary*, respectively. A typical question was, given a set of axioms $\mathcal{A}$, corresponding usually to some intuitively acceptable aspects of truth, what is the logic defined by $\mathcal{A}$, i.e. which formulas are provable from $\mathcal{A}$ in a Hilbert-like system.

One of the most important steps in the history of modal logic was inventing a formal semantics based on the notion of the so-called Kripke structures. Basically, a Kripke structure is a directed graph, called a *frame*, together with a valuation of propositional variables. Vertices of this graph are called *worlds*. For each world truth values of all propositional variables can be defined independently. In this semantics, $\Diamond\varphi$ means φ *is true in some world connected to the current world*; and $\Box\varphi$, equivalent to $\neg\Diamond\neg\varphi$, means φ *is true in all worlds connected to the current world.*

It appeared that there is a beautiful connection between syntactic and semantic approaches to modal logic [12]: logics defined by axioms can be equivalently defined by restricting classes of frames. E.g., the axiom $\Diamond\Diamond P \to \Diamond P$ (*if it is possible that P is possible, then P is possible*), is valid precisely in the class of transitive frames; the axiom $P \to \Diamond P$ (*if P is true, then P is possible*) – in the class of reflexive frames, $P \to \Box\Diamond P$ (*if P is true, then it is necessary*

* Partially supported by Polish Ministry of Science and Higher Education grant N N206 37133.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 264–275

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that P is possible) – in the class of symmetric frames, and the axiom $\Diamond P \to \Box\Diamond P$ (*if P is possible, then it is necessary that P is possible*) – in the class of Euclidean frames.

Thus we may think that every modal formula φ defines a class of frames, namely the class of those frames in which φ is valid. A formula φ is valid in a frame K if for any possible truth-assignment of propositional variables to the worlds of K, φ is true at every world. While this definition involves quantification over sets of worlds, many important classes of frames, in particular all the classes we mentioned above, can be defined by simple first-order formulas. For a given first-order sentence Φ over the signature consisting of a single binary symbol R we define $\mathcal{K}_\Phi$ to be the set of those frames which satisfy Φ.

In this paper we are interested in the satisfiability problem for modal logic over classes of frames definable by universal first-order formulas. The first result in this area was that there exists a universal first-order formula with equality Φ, such that the *global* satisfiability problem for modal logic over $\mathcal{K}_\Phi$ is undecidable [6]. By global satisfiability we mean the problem of determining if there exists a Kripke structure such that a given modal formula φ is true at every world of this structure. That result has been recently improved in [8] in two aspects: by removing equality and globalness. Namely, the authors exhibited a formula Φ' without equality, such that the standard, *local*, satisfiability problem for modal logic over $\mathcal{K}_{\Phi'}$ is undecidable.

The formula from [8] uses nine variables. A natural question arises, how many variables are necessary to obtain undecidability. Note that transitive, reflexive, symmetric, or equivalence frames are definable by formulas with just three variables. The satisfiability problem for modal logic over those classes is known to be decidable [9]. It appears however that there exists a universal first-order formula without equality with only three variables defining the class of frames over which satisfiability problem for modal logic is undecidable. Exhibiting such a formula is the first contribution of our paper.

▶ **Theorem 1.** *There exists a three-variable universal formula Γ', without equality, such that the local satisfiability problem for modal logic over $\mathcal{K}_{\Gamma'}$ is undecidable.*

Our formula, despite the fact that it uses much smaller number of variables, is also simpler than the formula from [8]. Actually, if we only want to show the undecidability of global satisfiability then we can use a formula Γ which is just a single, universally quantified clause consisting of six literals.

We emphasize that our result is optimal with respect to the number of variables. Indeed, if Φ is an arbitrary (not necessarily universal) first-order sentence with two variables, then the satisfiability problem for modal logic over $\mathcal{K}_\Phi$ can be reduced to the satisfiability problem for the two-variable fragment of first-order logic, FO^2, using the standard translation of modal logic into FO^2. The latter problem is known to be decidable [10, 4]. For details about the standard translation see e.g. [2].

Decidable classes of frames we mentioned earlier can be defined by three-variable first-order sentences even if we further restrict the language to universal Horn formulas, UHF. Universal Horn formulas were considered in [7], where a dichotomy result was proved, that the satisfiability problem for modal logic over the class of structures defined by an UHF formula (with an arbitrary number of variables) is either in NP or PSPACE-hard. In the same paper decidability is shown for a rich subclass of UHF, including in particular all formulas which imply reflexivity. However, the problem remained open for formulas involving variants of transitivity. The authors of [7] conjecture that the problem is decidable, and in PSPACE for all universal Horn formulas. Our second contribution is confirming this conjecture for the case of formulas with at most three-variables, UHF^3.

▶ **Theorem 2.** *Let Φ be a* UHF^3 *sentence. Then the local and the global satisfiability problems for modal logic over $\mathcal{K}_\Phi$ are decidable.*

This theorem extends the decidability results for the classes we mentioned earlier in this introduction, in particular for modal logics T, B, $\mathsf{K4}$, $\mathsf{S4}$, $\mathsf{S5}$. It also works for some interesting classes of frames, for which, up to our knowledge, decidability has not been established so far. An example is the class defined by $\forall xyz(xRy \wedge yRz \to zRx)$.

We provide a full classification of UHF^3 sentences, with respect to the complexity of satisfiability of modal logic over the classes of frames they define. It appears, that except for the trivial case of inconsistent formulas for which the problem is in P, local satisfiability is either NP-complete or PSPACE-complete, and global satisfiability is NP-complete, PSPACE-complete, or EXPTIME-complete.

2 Preliminaries

As we work with both first-order logic and modal logic we help the reader by distinguishing them in our notation: we denote first-order formulas with Greek capital letters, and modal formulas with Greek small letters. We assume that the reader is familiar with first-order and propositional logic.

Modal logic extends propositional logic with the operator $\Diamond$ and its dual $\Box$. Formulas of modal logic are interpreted in Kripke structures, which are triples of the form $\langle W, R, \pi\rangle$, where W is a set of worlds, $\langle W, R\rangle$ is a directed graph called a *frame*, and π is a function that assigns to each world a set of propositional variables which are true at this world. We say that a structure $\langle W, R, \pi\rangle$ is *based* on the frame $\langle W, R\rangle$.

The semantics of modal logic if defined recursively. A modal formula φ is (locally) *satisfied* in a world w of a model $\mathfrak{M} = \langle W, R, \pi\rangle$, denoted as $\mathfrak{M}, w \models \varphi$ if (i) φ is a variable and $\varphi \in \pi(w)$, (ii) $\varphi = \varphi_1 \vee \varphi_2$ and $\mathfrak{M}, w \models \varphi_1$ or $\mathfrak{M}, w \models \varphi_2$, (iii) $\varphi = \neg\varphi'$ and $\mathfrak{M}, w \not\models \varphi'$, or (iv) $\varphi = \Diamond\varphi'$ and there exists a world $v \in W$ such that $(w, v) \in R$ and $\mathfrak{M}, v \models \varphi'$. We abbreviate $\neg\Diamond\neg\varphi$ by $\Box\varphi$. By $|\varphi|$ we denote the length of φ measured as the total number of occurrences of propositional variables. We say that a formula φ is *globally* satisfied in $\mathfrak{M}$, denoted as $\mathfrak{M} \models \varphi$, if for all worlds w of $\mathfrak{M}$, we have $\mathfrak{M}, w \models \varphi$.

For a given class of frames $\mathcal{K}$, we say that a formula φ is *locally* (resp. *globally*) *$\mathcal{K}$-satisfiable* if there exists a frame $K \in \mathcal{K}$, a structure $\mathfrak{M}$ based on K, and a world $w \in W$ such that $\mathfrak{M}, w \models \varphi$ (resp. $\mathfrak{M} \models \varphi$). We define the *local* (resp. *global*) *satisfiability problem* $\mathcal{K}$-SAT (resp. global $\mathcal{K}$-SAT) as follows. For a given modal formula, is this formula locally (resp. globally) $\mathcal{K}$-satisfiable?

For a given formula φ, a Kripke structure $\mathfrak{M}$, and a world $w \in W$ we define the *type* of w (with respect to φ) in $\mathfrak{M}$ as $tp^\varphi_\mathfrak{M}(w) = \{\psi : \mathfrak{M}, w \models \psi \text{ and } \psi \text{ is subformula of } \varphi\}$. We write $tp_\mathfrak{M}(w)$ if the formula is clear from the context. Note that $|tp^\varphi_\mathfrak{M}(w)| < |\varphi|$.

The set of *universal Horn formulas* with three variables without equality, UHF^3, is defined as the set of those Φ which are of the form $\forall xyz.\Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_i$, where each Φ_i is a Horn clause. A Horn clause is a disjunction of literals of which at most one is positive. We usually present Horn clauses as implications. For example, the formula $\forall xyz.(xRy \wedge yRz \Rightarrow xRz) \wedge (xRx \Rightarrow \bot)$ defines the set of transitive and irreflexive frames. We often skip the quantifiers and represent such formulas as sets of clauses, e.g.: $\{xRy \wedge yRz \Rightarrow xRz, xRx \Rightarrow \bot\}$. We assume without loss of generality that each Horn clause is of the form $\Psi \Rightarrow \bot$, $\Psi \Rightarrow xRx$, or $\Psi \Rightarrow xRy$. We define $\Psi(v_1, v_2, v_3)$ as the instantiation of Ψ with $x = v_1$, $y = v_2$, and $z = v_3$, e.g. $(xRy \wedge yRz)(a, b, c) = aRb \wedge bRc$. We denote by Φ^p the set of the clauses of Φ containing a positive literal, i.e. all clauses of Φ except those of the form $\Psi \Rightarrow \bot$.

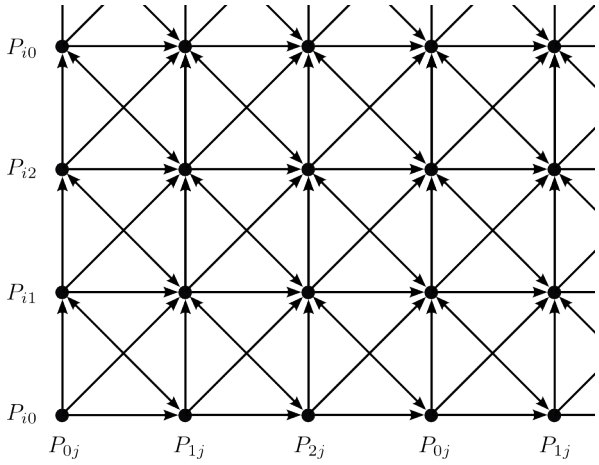


Figure 1 The structure $\mathfrak{G}_{\mathbb{N}}$. Its universe is $\mathbb{N} \times \mathbb{N}$. Reflexive arrows are omitted for clarity.

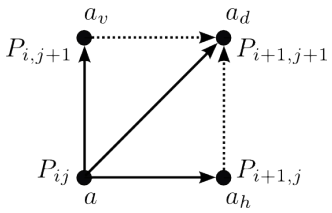


Figure 2 Completing the grid. Dotted arrows are enforced by Γ and τ.

3 Undecidability

In this section we work with signatures consisting of a single binary symbol R, and a number of unary symbols, including P_{ij}, for $0 \leq i, j \leq 2$. Structures over such signatures can be naturally viewed as Kripke structures in which R is the accessibility relation, and unary relations describe valuations of propositional variables. To simplify our notation we assume that subscripts in P_{ij} are always taken modulo 3, e.g. if $i = 2, j = 2$, then $P_{i+1,j+1}$ denotes P_{00}.

Let

$$\Gamma = \forall xyz.\neg xRy \vee yRx \vee \neg xRz \vee zRx \vee yRz \vee zRy.$$

First, we prove that global $\mathcal{K}_\Gamma$-SAT is undecidable. Then we use the trick from [8] and show that also local $\mathcal{K}_{\Gamma'}$-SAT is undecidable, for Γ' being a modification of Γ, using still only three variables.

3.1 General idea

Note that Γ can be rewritten as $\forall xyz.(xRy \wedge \neg yRx \wedge xRz \wedge \neg zRx) \rightarrow (yRz \vee zRy)$, i.e. it says, that if there are one-way connections from a world x to worlds y, z, then there is also a connection (not necessarily one-way) between y and z. The structure $\mathfrak{G}_{\mathbb{N}}$ illustrated in Fig. 1 (we assume that this structure is reflexive) is a model of Γ. Note that it is important that some connections are two-way. In $\mathfrak{G}_{\mathbb{N}}$ we can define the horizontal adjacency relation by the following formula with free variables x, y: $\bigvee_{ij}(P_{ij}x \wedge P_{i+1,j}y \wedge xRy)$. Analogously, we define the vertical adjacency: $\bigvee_{ij}(P_{ij}x \wedge P_{i,j+1}y \wedge xRy)$. $\mathfrak{G}_{\mathbb{N}}$ can be now viewed as an expansion of the standard grid on $\mathbb{N} \times \mathbb{N}$.

To get the undecidability we construct a modal formula τ, capturing some properties of $\mathfrak{G}_{\mathbb{N}}$, such that any model $\mathfrak{M} \models \tau$ from $\mathcal{K}_\Gamma$ locally looks like a grid. Namely, τ says that every element satisfying P_{ij} has three R-successors: one in $P_{i+1,j}$, one in $P_{i,j+1}$, and one in $P_{i+1,j+1}$, and forbids connections from $P_{i+1,j+1}$ to $P_{i,j+1}$, $P_{i+1,j}$, and P_{ij}. If we consider now any element a in a model, we see that τ enforces the existence of its horizontal successor a_h, its vertical successor a_v and its upper-right diagonal successor a_d (see Fig. 2). By τ, the connections to these successors are one-way, so we need, by Γ, connections between a_h and

a_d, and a_v and a_d. Again, by τ, these connections has to go from a_h to a_d, and from a_v to a_d, so a_d is indeed a horizontal successor of a_v, and a vertical successor of a_h.

Below we present a more detailed proof covering also the case of finite satisfiability, i.e. satisfiability in the class of finite models. The technique we employ is quite standard. It is similar e.g. to the technique used in [11].

3.2 Domino systems

In the proof we use some well known results on domino systems.

▶ **Definition 3.** A *domino system* is a tuple $\mathcal{D} = (D, D_H, D_V)$, where D is a set of domino pieces and $D_H, D_V \subseteq D \times D$ are binary relations specifying admissible horizontal and vertical adjacencies. We say that $\mathcal{D}$ *tiles* $\mathbb{N} \times \mathbb{N}$ if there exists a function $t : \mathbb{N} \times \mathbb{N} \to \mathcal{D}$ such that $\forall i, j \in \mathbb{N}$ we have $(t(i,j), t(i+1,j)) \in D_H$ and $(t(i,j), t(i,j+1)) \in D_V$. Similarly, $\mathcal{D}$ tiles $\mathbb{Z}_k \times \mathbb{Z}_l$, for $k, l \in \mathbb{N}$, if there exists $t : \mathbb{Z}_k \times \mathbb{Z}_l \to \mathcal{D}$ such that $(t(i,j), t(i+1 \mod k, j)) \in D_H$ and $(t(i,j), t(i, j+1 \mod l)) \in D_V$.

The following lemma comes from [1, 5].

▶ **Lemma 4.** *The following problems are undecidable:*

(i) For a given domino system $\mathcal{D}$ determine if $\mathcal{D}$ tiles $\mathbb{N} \times \mathbb{N}$.
(ii) For a given domino system $\mathcal{D}$ determine if there exists $k \in \mathbb{N}$ such that $\mathcal{D}$ tiles $\mathbb{Z}_k \times \mathbb{Z}_k$.

3.3 Grid definition

We capture some properties of $\mathfrak{G}_{\mathbb{N}}$ by a modal formula τ.

$$\tau = \tau_0 \wedge \bigwedge_{0 \leq i,j \leq 2} (\tau_{ij}^{\Diamond} \wedge \tau_{ij}^{\Box}),$$

where τ_0 says that each element satisfies one of P_{ij}, $\tau_{ij}^{\Diamond}$ ensure that all elements have appropriate horizontal, vertical and upper-right diagonal successors, and $\tau_{ij}^{\Box}$ forbid reversing the horizontal, vertical and upper-right diagonal arrows.

$$\tau_{ij}^{\Diamond} = P_{ij} \to (\Diamond P_{i+1,j} \wedge \Diamond P_{i,j+1} \wedge \Diamond P_{i+1,j+1}),$$

$$\tau_{ij}^{\Box} = P_{ij} \to \Box(\neg P_{i-1,j} \wedge \neg P_{i,j-1} \wedge \neg P_{i-1,j-1}).$$

Note that $\tau_{ij}^{\Box}$ allow for reflexive edges.

3.4 Domino encoding

We encode an instance of the domino problem by a modal formula in a standard way. For a given domino system $\mathcal{D} = (D, D_H, D_V)$ we define

$$\lambda^{\mathcal{D}} = \lambda_0 \wedge \bigwedge_{0 \leq i,j \leq 2} (\lambda_{ij}^H \wedge \lambda_{ij}^V).$$

For every $d \in D$ we use a fresh propositional letter P_d. λ_0 says that each world contains a domino piece, λ_{ij}^H and λ_{ij}^V say that pairs of elements satisfying horizontal and vertical adjacency relations respect D_H and D_V, respectively.

$$\lambda_{ij}^H = \bigwedge_{d \in D} ((P_d \wedge P_{ij}) \to \Box(P_{i+1,j} \to \bigvee_{d' : (d,d') \in D_H} P_{d'})),$$

$$\lambda^V_{ij} = \bigwedge_{d\in D}((P_d \wedge P_{ij}) \to \Box(P_{i,j+1} \to \bigvee_{d':(d,d')\in D_V} P_{d'})).$$

The following lemma establishes the undecidability of the global satisfiability and the global finite satisfiability problems for modal logic over $\mathcal{K}_\Gamma$.

▶ **Lemma 5.** *Let $\mathcal{D}$ be a domino system.*

(i) $\mathcal{D}$ tiles $\mathbb{N}\times\mathbb{N}$ iff there exists $\mathfrak{M}\in\mathcal{K}_\Gamma$ such that $\mathfrak{M}\models\tau\wedge\lambda^{\mathcal{D}}$.
(ii) $\mathcal{D}$ tiles some $\mathbb{Z}_k\times\mathbb{Z}_k$ iff there exists a finite $\mathfrak{M}\in\mathcal{K}_\Gamma$ such that $\mathfrak{M}\models\tau\wedge\lambda^{\mathcal{D}}$.

Proof. As in the case of symbols P_{ij}, when referring to $\tau^{\Box}_{ij}$ or $\tau^{\Diamond}_{ij}$ we assume that subscripts are taken modulo 3.

Part (i), $\Rightarrow$ Let t be a tiling of $\mathbb{N}\times\mathbb{N}$. We construct $\mathfrak{M}$ by expanding $\mathfrak{G}_{\mathbb{N}}$ in such a way that for every $i,j\in\mathbb{N}$ the element (i,j) satisfies $P_{t(i,j)}$. It is readily checked that $\mathfrak{M}$ is as required.

Part (i), $\Leftarrow$ We explain how to construct a function $f:\mathbb{N}\times\mathbb{N}\to M$, such that for every $i,j\in\mathbb{N}$: (a) $\mathfrak{M}\models P_{ij}(f(i,j))$, (b) $\mathfrak{M}\models f(i,j)Rf(i+1,j)$, (c) $\mathfrak{M}\models f(i,j)Rf(i,j+1)$.

First we show how to define f on $\mathbb{N}\times\{0\}$. Let $f(0,0)=c$ for an arbitrary element c of M satisfying P_{00}. Such c exists owing to τ_0 and $\tau^{\Diamond}_{ij}$. Assume that for some $i>0$ we have defined $f(i-1,0)=a$, and let a_h be an R-successor of a satisfying P_{i0}. Such a_h exists owing to $\tau^{\Diamond}_{i-1,0}$. Define $f(i,0)=a_h$.

Assume now that f is defined for $\mathbb{N}\times[0,\ldots,j-1]$ for some $j>0$. We extend this definition to $\mathbb{N}\times\{j\}$. Let $f(0,j-1)=a$. By the inductive assumption a satisfies $P_{0,j-1}$. Choose a_v to be an R-successor of a satisfying P_{0j}. Such a_v exists by $\tau^{\Diamond}_{0,j-1}$. Set $f(0,j)=a_v$.

Assume that we have defined $f(i-1,j-1)=a$, $f(i-1,j)=a_v$, and $f(i,j-1)=a_h$. By the inductive assumptions $\mathfrak{M}\models P_{i-1,j-1}(a)\wedge P_{i-1,j}(a_v)\wedge P_{i,j-1}(a_h)\wedge aRa_h\wedge aRa_v$. Choose a_d to be an R-successor of a satisfying P_{ij}. Such a_d exists by $\tau^{\Diamond}_{i-1,j-1}$. By $\tau^{\Box}_{ij}$, a_h, a_v and a_d cannot be connected to a, so Γ enforces R-connections between a_h and a_d, and between a_v and a_d. Since τ^{A}_{ij} forbids connection from a_d to a_h, and from a_d to a_v, it has to be that $\mathfrak{M}\models a_hRa_d\wedge a_vRa_d$. This finishes definition of f with the desired properties.

We define a tiling $t:\mathbb{N}\times\mathbb{N}$ by setting $t(i,j)=d$ for such d that $f(i,j)$ satisfies P_d (there is at least one such d owing to λ_0). Properties (a), (b), (c) of f and the formulas $\lambda^H_{ij},\lambda^V_{ij}$ imply that t is a correct tiling.

Part (ii) $\Rightarrow$ Let $l=3k$ for some $k\in\mathbb{Z}$. We define $\mathfrak{G}_l$ to be the quotient of $\mathfrak{G}_{\mathbb{N}}$ by the relation $\approx$: $(i,j)\approx(i',j')$ iff $i\equiv i' \mod l$ and $j\equiv j' \mod l$. $\mathfrak{G}_l$ can be seen as an expansion of the standard grid on $\mathbb{Z}_l\times\mathbb{Z}_l$ torus. It is readily checked that for every $k\in\mathbb{N}$ we have $\mathfrak{G}_{3k}\models\Gamma$ and $\mathfrak{G}_{3k}\models\tau$.

If $\mathcal{D}$ tiles $\mathbb{Z}_k\times\mathbb{Z}_k$ then it also tiles $\mathbb{Z}_{3k}\times\mathbb{Z}_{3k}$. Let t be a tiling of $\mathbb{Z}_{3k}\times\mathbb{Z}_{3k}$. We construct $\mathfrak{M}$ by expanding $\mathfrak{G}_{3k}$ in such a way that for every $i,j\in\mathbb{Z}_{3k}$ the element (i,j) satisfies $P_{t(i,j)}$. Again, checking that $\mathfrak{M}$ is as required is straightforward.

Part (ii) $\Leftarrow$ We want to define for some $k,l\in\mathbb{Z}$ a function $f:\mathbb{Z}_k\times\mathbb{Z}_l\to M$ satisfying: (a) $\mathfrak{M}\models P_{ij}(f(i,j))$, (b) $\mathfrak{M}\models f(i,j)Rf(i+1 \mod k,j)$, (c) $\mathfrak{M}\models f(i,j)Rf(i,j+1 \mod l)$.

We define f as a partial function on $\mathbb{N}\times\mathbb{N}$ and then restrict it to an appropriate domain. We first define f on $\mathbb{N}\times\{0\}$, exactly as in the proof of Part (i), $\Leftarrow$. Since $\mathfrak{M}$ is finite this time, it has to be that $f(k,0)=f(k',0)$ for some $k>k'$. To simplify the presentation we assume $k'=0$, but this assumption is not relevant. Observe that for $i\in[0,k)$ we have $\mathfrak{M}\models f(i,0)Rf(i+1 \mod k,0)$. We extend the definition of f to $[0,k)\times\mathbb{N}$ inductively. Assume that f is defined on $[0,k)\times\{0,\ldots,j-1\}$. We define it on $[0,k)\times\{j\}$. For each $i\in[0,k)$ we

find an element a^i_d in M such that $\mathfrak{M} \models P_{i+1,j}(a^i_d) \wedge f(i,j-1)Ra^i_d$. Such a^i_d exists owing to $\tau^{\Diamond}_{i,j-1}$. We set $f(i+1 \mod k, j) = a^i_d$. Now Γ and formulas of the type $\tau^{\Box}$ enforce for all $i \in [0,k)$ that $\mathfrak{M} \models f(i,j-1)Rf(i,j)$, and $\mathfrak{M} \models f(i,j)Rf(i+1 \mod k, j)$.

Finiteness of $\mathfrak{M}$ implies now that for some $l > l'$ we have $f{\restriction}[0,k) \times \{l\} = f{\restriction}[0,k) \times \{l'\}$. Again for simplicity we assume that $l' = 0$. Observe that at this moment f is as desired on $\mathbb{Z}_k \times \mathbb{Z}_l$. We define a tiling $t : \mathbb{Z}_k \times \mathbb{Z}_l$ by setting $t(i,j) = d$ for such d that $f(i,j)$ satisfies P_d (there is at least one such d owing to λ_0). Properties (a), (b), (c) of f and the formulas λ^H_{ij} and λ^V_{ij} imply that t is a correct tiling of $\mathbb{Z}_k \times \mathbb{Z}_l$. This implies that there exists also a correct tiling of $\mathbb{Z}_m \times \mathbb{Z}_m$ for $m = gcd(k,l)$.

◀

3.5 Local satisfiability

Observe that our proof of the undecidability of global satisfiability over $\mathcal{K}_\Gamma$ works for the subclass of reflexive models. This allows us to use the trick from [8] to cover also the case of local satisfiability. We enforce by a modal formula the existence of an irreflexive world and, by a first-order formula, we make it connected to all reflexive worlds. Such a *universal world* can be then used to reach all relevant elements in the model. The class of structures is defined by a formula Γ', which says that each world with an incoming edge is reflexive and has an incoming edge from all irreflexive worlds, and enforces Γ for all reflexive worlds:

$$\Gamma' = \forall xyz.((xRy \wedge \neg zRz) \to (yRy \wedge zRy)) \wedge \\ ((xRx \wedge yRy \wedge zRz) \to (\neg xRy \vee yRx \vee \neg xRz \vee zRx \vee yRz \vee zRy)).$$

In the modal formula we use a fresh symbol P_U to distinguish an irreflexive world. Now, for a given domino system $\mathcal{D}$ we can show that $P_U \wedge \Box\neg P_U \wedge \Diamond\top \wedge \Box(\tau \wedge \lambda^D)$ is locally (finitely) satisfiable over $\mathcal{K}_{\Gamma'}$ iff $\mathcal{D}$ covers $\mathbb{N} \times \mathbb{N}$ (some $\mathbb{Z}_k \times \mathbb{Z}_k$). This proves Theorem 1.

See subsection 5.6 of [8] for details of the outlined trick.

4 Decidability

In this section, we prove Theorem 2. The general idea of the proof is standard: we are going to show that for every UHF^3 formula Φ and every modal formula φ, if φ is $\mathcal{K}_\Phi$-satisfiable then it is also $\mathcal{K}_\Phi$-satisfiable in a "nice" model.

We start from an arbitrary model $\mathfrak{M} \models \varphi$ based on a frame from $\mathcal{K}_\Phi$ and unravel it into a model $\mathfrak{M}_0$ whose frame is a tree with the degree of its nodes bounded by $|\varphi|$. Clearly the frame of $\mathfrak{M}_0$ is not necessarily a member of $\mathcal{K}_\Phi$. In the next step we add to $\mathfrak{M}_0$ the edges implied by the Horn clauses of Φ. This is performed in countably many stages, until the least fixed point is reached. We observe that the resulting structure, $\mathfrak{M}_\infty$, is still a model of φ, and its frame belongs to $\mathcal{K}_\Phi$.

Then we show that every model which can be obtained in the described way falls into one of the four classes, which we call the class of semi-trees, transitive-trees, clique-unions, and tripartitions.[1] Moreover, for a given UHF^3 formula Φ there exists a single class of models, such that every $\mathcal{K}_\Phi$-satisfiable modal formula φ has a model from this class.

Finally, we argue that for a given modal formula φ, checking if it has a model from one of our four classes is decidable. If φ is $\mathcal{K}_\Phi$-satisfiable in a clique-union or in a tripartition it

[1] We choose such names for simplicity. In fact, in transitive trees transitivity may fail near the end of a path, and clique-unions may have *heads* and *tails*. See Definition 7.

can be shown that it is also $\mathcal{K}_\Phi$-satisfiable in a clique-union or a tripartition of polynomially bounded size, so we can simply guess such a small model and verify it; if φ is $\mathcal{K}_\Phi$-satisfiable in a semi-tree or in a transitive-tree then we use some adaptations of the standard techniques for satisfiability of modal logics over the class of all frames, and over the class of transitive frames, respectively.

4.1 Minimal tree-based models

We say that an edge (w_1, w_2) is a *consequence* of Φ in $\langle W, R\rangle$ if for some $w_3 \in W$ and $\Psi_1 \Rightarrow \Psi_2 \in \Phi$ we have $R \models \Psi_1(w_1, w_2, w_3)$, and $\Psi_2(w_1, w_2, w_3) = w_1 R w_2$. We define the *consequence operator* as follows.

$$\textsc{Cons}_{\Phi,W}(R) = R \cup \{(w_1, w_2) : (w_1, w_2) \text{ is a consequence of } \Phi \text{ in } \langle W, R\rangle\}$$

We are going to use this operator in stages, starting from a tree and adding edges required by Φ. We define the *closure operator* as the least fixed-point of *Cons*:

$$\textsc{Closure}_{\Phi,W}(R) = \bigcup_{i>0} \textsc{Cons}^i_{\Phi,W}(R)$$

For a tree $\mathcal{T} = \langle W, R\rangle$, we now define the *minimal* T*-based model of* Φ as $\mathfrak{C}_\Phi(\mathcal{T}) = \langle W, Closure_{\Phi,W}(R)\rangle$. Note that $\mathfrak{C}_\Phi(\mathcal{T})$ is the smallest model of Φ^p containing all edges from R.

▶ **Lemma 6.** *Let φ be a modal formula and let $\Phi \in \mathsf{UHF}^3$. If φ is $\mathcal{K}_\Phi$-satisfiable, then there exists a tree $\mathcal{T}$ in which the degree of its nodes is bounded by $|\varphi|$, such that φ has a model based on the frame $\mathfrak{C}_\Phi(\mathcal{T})$.*

Proof. Let $\mathfrak{M} = \langle W, R, \pi\rangle$, $u_0 \in W$ be such that $\mathfrak{M} \models \Phi$ and $\mathfrak{M}, u_0 \models \varphi$.

We construct $\mathfrak{M}_0 = \langle W_0, R_0, \pi_0\rangle$ by an unraveling of $\mathfrak{M}$ as follows. W_0 is a subset of the set of finite sequences of elements of W. We define W_0 and R_0 inductively. Initially, we put $(u_0) \in W_0$. Assume that $(u_0, \ldots, u_k) \in W_0$. Let $\Diamond\psi_1, \ldots, \Diamond\psi_s$ be all the formulas of the form $\Diamond\psi$ from $tp_{\mathfrak{M}}(u_k)$. There exist $u^1_{k+1}, \ldots, u^s_{k+1} \in W$, such that for every $i \in \{1, \ldots, s\}$ we have $\mathfrak{M} \models u_k R u^i_{k+1}$ and $\psi_i \in tp_{\mathfrak{M}}(u^i_{k+1})$. For each such i we put $(u_0, \ldots, u_k, u^i_{k+1})$ into W_0 and add $((u_0, \ldots, u_k), (u_0, \ldots, u_k, u^i_{k+1}))$ to R_0. We define π_0 as $\pi_0((u_0, \ldots, u_k)) = \pi(u_k)$. Observe that $\mathcal{M}_0 = \langle W_0, R_0\rangle$ is a tree in which the degree of the nodes is bounded by $|\varphi|$.

Let $f : W_0 \to W$ be defined as $f((u_0, \ldots, u_k)) = u_k$. By a straightforward induction the reader may verify that, for every $\vec{u} \in W_0$ we have $tp_{\mathfrak{M}_0}(\vec{u}) = tp_{\mathfrak{M}}(f(\vec{u}))$. This implies that $\mathfrak{M}_0, (u_0) \models \varphi$.

Now, in countably many stages we add to $\mathcal{M}_0$ the edges implied by Φ. We define a sequence of frames $(\mathcal{M}_i)_{i>0}$ and models $(\mathfrak{M}_i)_{i>0}$ sharing the same universe W_0 and mapping π_0. For $K > 0$ let $\mathcal{M}_K = \langle W_0, \textsc{Cons}^K_{\Phi,W_0}(R_0)\rangle$, $\mathfrak{M}_K = \langle \mathcal{M}_K, \pi_0\rangle$. Let $\mathfrak{M}_\infty$ be the natural limit $\mathfrak{M}_\infty = \langle \mathfrak{C}_\Phi(\mathcal{M}_0), \pi_0\rangle$.

We show by induction over K, that for each $\vec{u}_1, \vec{u}_2 \in W_0$ if $\mathfrak{M}_K \models \vec{u}_1 R \vec{u}_2$, then $\mathfrak{M} \models f(\vec{u}_1) R f(\vec{u}_2)$. It follows that for each $\vec{u}_1, \vec{u}_2 \in W_0$ if $\mathfrak{M}_\infty \models \vec{u}_1 R \vec{u}_2$, then $\mathfrak{M} \models f(\vec{u}_1) R f(\vec{u}_2)$. For $K = 0$ the conclusion is a straightforward consequence of the definition of $\mathfrak{M}_0$. Assume that $\mathfrak{M}_K$ satisfies the inductive hypothesis. For each $\vec{u}_1, \vec{u}_2 \in W_0$, if $\mathfrak{M}_{K+1} \models \vec{u}_1 R \vec{u}_2$, then either $\mathfrak{M}_K \models \vec{u}_1 R \vec{u}_2$ and by the inductive assumption $\mathfrak{M} \models f(\vec{u}_1) R f(\vec{u}_2)$, or for some $\vec{u_3} \in W_0$ and $\Psi_1 \Rightarrow \Psi_2 \in \Phi$, we have $\mathfrak{M}_K \models \Psi_1(\vec{u_1}, \vec{u_2}, \vec{u_3})$, and $\Psi_2(\vec{u_1}, \vec{u_2}, \vec{u_3}) = \vec{u_1} R \vec{u_2}$. In this case, $\mathfrak{M}_K \models \Psi_1(\vec{u_1}, \vec{u_2}, \vec{u_3})$ implies by the inductive assumption that $\mathfrak{M} \models \Psi_1(f(\vec{u_1}), f(\vec{u_2}), f(\vec{u_3}))$. Since $\mathfrak{M} \models \Psi_1 \Rightarrow \Psi_2$, we have $\mathfrak{M} \models f(\vec{u}_1) R f(\vec{u}_2)$.

Let $\mathfrak{M}_\infty = \langle W_0, R_\infty, \pi_0\rangle$. The structures $\mathfrak{M}_0$ and $\mathfrak{M}_\infty$ have the same carrier and $R_0 \subseteq R_\infty$. We show that for each $\vec{u} \in W_0$ we have $tp_{\mathfrak{M}_\infty}(\vec{u}) = tp_{\mathfrak{M}_0}(\vec{u})$. It implies that

$\mathfrak{M}_\infty, (u_0) \models \varphi$. Since the labeling of the worlds is the same, it is enough to show that in $\mathfrak{M}_0$ and $\mathfrak{M}_\infty$ each world is connected with the worlds that satisfy the same subformulas. We show that by induction.

Clearly, for every edge $(\vec{u}, \vec{v})$ from $R_\infty \setminus R_0$ and a subformula $\Diamond\psi$ of φ, if a world $\vec{v}$ satisfies ψ in $\mathfrak{M}_\infty$, then by the inductive assumption we have that $\psi \in tp_{\mathfrak{M}_0}(\vec{v}) = tp_{\mathfrak{M}}(f(\vec{v}))$, and since $\mathfrak{M} \models f(\vec{u})Rf(\vec{v})$ we have that $\Diamond\psi \in tp_{\mathfrak{M}}(f(\vec{u})) = tp_{\mathfrak{M}_0}(\vec{u})$. See the full version of this paper for a detailed proof.

Finally, we have to prove that $\mathfrak{C}_\Phi(\mathcal{M}_0) \models \Phi$. By definition $\mathfrak{C}_\Phi(\mathcal{M}_0)$ satisfies every $\Psi_1 \Rightarrow \Psi_2 \in \Phi^p$. Suppose that $\mathfrak{C}_\Phi(\mathcal{M}_0)$ does not satisfy $\Psi \Rightarrow \bot \in \Phi$. For some $\vec{w_1}, \vec{w_2}, \vec{w_3}$ we have $\mathfrak{C}_\Phi(\mathcal{M}_0) \models \Psi(\vec{w_1}, \vec{w_2}, \vec{w_3})$, but then $\mathfrak{M} \models \Psi(f(\vec{w_1}), f(\vec{w_2}), f(\vec{w_3}))$. This contradicts the assumption that $\mathfrak{M} \models \Phi$. ◀

4.2 Catalogue of models

A well known result shows that every satisfiable modal formula is satisfied in a finite tree. This *tree-model property* is crucial for the robust decidability of modal logics. Standard restrictions of classes of frames lead to similar results, stating that some "nice" models exist for all satisfiable formulas. For example, every formula satisfiable over transitive structures has a model which is a transitive tree.

Here we generalize those results. We introduce four classes of models and show that for each formula Φ all formulas satisfiable over $\mathcal{K}_\Phi$ have models in one of those classes.

▶ **Definition 7.** We say that a graph $\langle W, R\rangle$ is

- a *semi-tree* if and only if there exists $R_0 \subseteq R$ such that $\langle W, R_0\rangle$ is a tree and R is contained in the reflexive, symmetric closure of R_0.
- a *transitive-tree* if and only if there exists $R_0 \subseteq R$ such that $\langle W, R_0\rangle$ is a tree, R is contained in the reflexive, transitive closure of R_0, and for each directed path $(u_0, u_1, \ldots, u_k)$ in $\langle W, R\rangle$ and each $2 \leq i \leq j \leq k-2$ we have an edge from u_i to u_j.
- a *tripartition* if and only if W can be partitioned into three independent sets I_1, I_2, I_3 such that for $d \in I_i$ and $e \in I_j$ we have that $dRe \iff j = i+1 \mod 3$.
- a *clique-union* if and only if W can be partitioned into $Head, Tails, C_1, \ldots, C_k$, where $C_1, \ldots, C_k$ are disjoint cliques, $Heads$ is a semi-tree of height at most 2, $Tails$ is a forest of semi-trees of height at most 2, and there are no edges from $C_1 \cup \ldots \cup C_k$ to $Head$ and from $Tails$ to $Head \cup C_1 \cup \ldots \cup C_k$.

▶ **Lemma 8.** *Let $\Phi \in \mathsf{UHF}^3$. One of the following conditions holds:*

- *For each tree $\mathcal{T}$, the structure $\mathfrak{C}_\Phi(\mathcal{T})$ is a semi-tree.*
- *For each tree $\mathcal{T}$, the structure $\mathfrak{C}_\Phi(\mathcal{T})$ is a transitive tree.*
- *For each tree $\mathcal{T}$, the structure $\mathfrak{C}_\Phi(\mathcal{T})$ is a tripartition.*
- *For each tree $\mathcal{T}$, the structure $\mathfrak{C}_\Phi(\mathcal{T})$ is a clique-union.*

This lemma, together with Lemma 6 imply that for every $\Phi \in \mathsf{UHF}^3$ modal logic has one of: *semi-tree model property, transitive tree model property, tripartite model property, clique-union model property* over $\mathcal{K}_\Phi$, i.e. every satisfiable formula has a model in one particular class. The proof of Lemma 8 starts from the analysis of the possible shapes of $\mathfrak{C}_\Phi(\mathcal{I})$, for the four-element tree $\mathcal{I}$ consisting of a root a, its two children b, d and a child c of b. It appears that studying what happens on this simple tree allows to see what can happen on arbitrary trees. The whole proof goes by a careful analysis of cases. Details are given in the full version of this paper. Here we only show some examples.

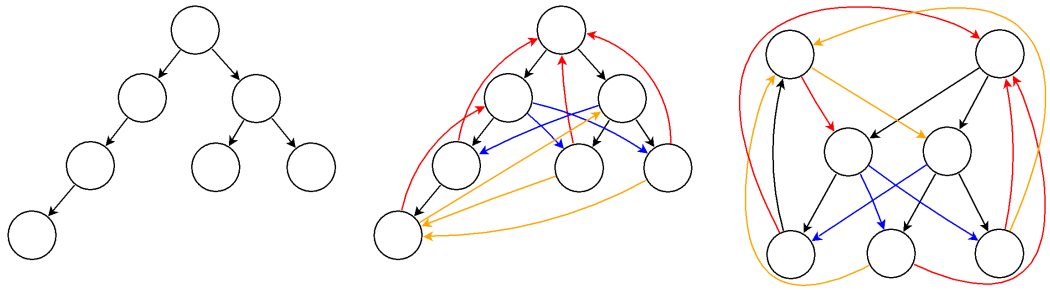


Figure 3 A closure for $\Phi = \{xRz \wedge zRy \Rightarrow yRx\}$ – three independent sets.

▶ **Example 9.** Consider the formula $\Phi = \{xRz \wedge zRy \Rightarrow yRx\}$ and the tree $\mathcal{T} = \langle W, R\rangle$ at the left side of Fig. 3. In the middle we present $\mathfrak{C}_\Phi(\mathcal{T})$ – red edges belong to $\textsc{Cons}_\Phi(R)$, blue to $\textsc{Cons}^2_\Phi(R)$, and yellow to $\textsc{Cons}^3_\Phi(R)$. Observe that each world from the level i of the tree is connected to all the worlds from the levels $i+1$ and $i-2$. On the right side of the figure we redraw the structure in a way underlining the partition into the three independent sets.

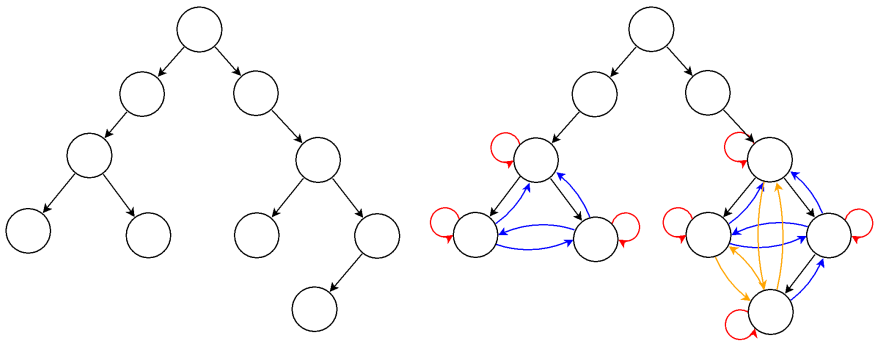


Figure 4 A closure for $\Phi = \{xRz \wedge zRy \Rightarrow yRy, xRx \wedge xRy \wedge xRz \Rightarrow yRz\}$ – a clique-union ($Tails = \emptyset$ in this example).

▶ **Example 10.** Consider the formula $\Phi = \{\varphi_1, \varphi_2\}$, where $\varphi_1 = xRz \wedge zRy \Rightarrow yRy$ and $\varphi_2 = xRx \wedge xRy \wedge xRz \Rightarrow yRz$, and the tree at the left side of Fig. 4. The formula φ_1 enforces the following property: each world that has a predecessor that has a predecessor is reflexive. The formula φ_2 makes the relation R Euclidean except for the non-reflexive worlds. As you can see at the right side of the figure, the fragment on which R is Euclidean collapses into a clique.

▶ **Example 11.** Consider the formula $\Phi = \{\varphi_1, \varphi_2\}$, where $\varphi_1 = xRy \wedge yRz \Rightarrow yRx$ and $\varphi_2 = xRy \wedge yRx \Rightarrow xRz$, and the tree at the left side of Fig. 5. The formula φ_1 enforces R to be symmetric, except for the edges that go to the worlds with no successors. The formula φ_2 enforces connections from each world symmetrically connected to some other world to all other worlds. As you can see at the right side of the figure, all worlds except for the leaves of the tree form a clique.

4.3 Decidability procedures and complexity

In this subsection we sketch procedures deciding satisfiability of modal logics over classes definable by UHF^3, and discuss the complexity. We exclude from our considerations formulas

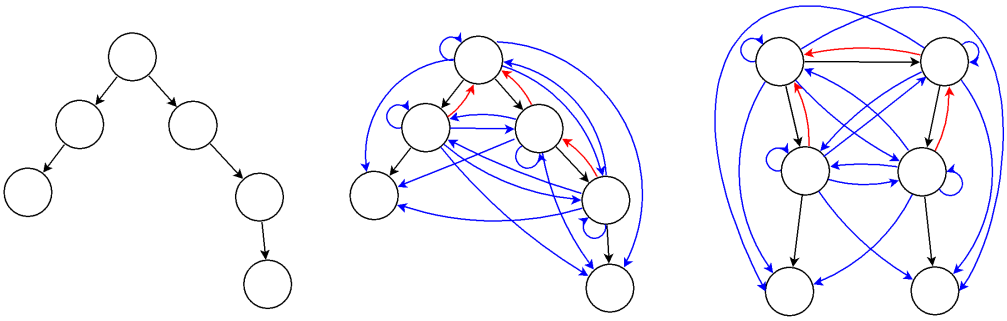


Figure 5 A closure for $\Phi = \{xRy \wedge yRz \Rightarrow yRx, xRy \wedge yRx \Rightarrow xRz\}$ – a clique with tails.

allowing only for paths of lenght bounded by a constant, e.g. $xRy \wedge yRz \to \bot$. Clearly, the satisfiability problem over classes of frames defined by such formulas is NP-complete.

Tripartitions and clique unions. It appears that in these two cases we can prove the following *polynomial model property*.

▶ **Lemma 12.** *For a given* UHF^3 *formula* Φ *and a modal formula* φ *if* φ *has a model in* $\mathcal{K}_\Phi$ *which is a tripartition or a clique union then it has a finite model of the same kind of size polynomially bounded by* $|\varphi|$.

Consider first the case of tripartitions. For every subformula $\Diamond\psi$ of φ, and every class of the partition I_i, if ψ is true at some elements of I_i, then we mark one such element. We also mark an element satisfying φ. We remove all unmarked elements. Since for a pair of classes of the partition they are either not connected or connected universally this procedure does not affect types of elements, so they still satisfy the same subformulas of φ.

The case of clique-unions is slightly more complicated. Recall that models from this class except cliques may also contain heads and tails, which cause that sometimes for a subformula $\Diamond\psi$ of φ we need more than one element satisfying ψ in a clique. However, the number of such elements may be bounded polynomially in $|\varphi|$. Similarly, we can also bound the number of cliques and tails. Technical details can be found in the full version of this paper.

In both cases the outlined arguments work for both local and global satisfiability. The decision procedure is to guess for a given formula φ a model of polynomial size and verify it. This establishes NP-upper bound. The matching lower bound follows from a trivial reduction from the boolean satisfiability problem.

Semi-trees. Here we can use standard approaches to satisfiability of modal logic over the class of all frames. In the case of local satisfiability we can bound the depth of tree-models and the degree of their nodes linearly in $|\varphi|$ and then check the existence of such models in a depth-first search manner in PSpace. (see e.g. [9][2]). The lower bound comes from the standard reduction of QBF (see also [9]).

In the case of global satisfiability we can enforce models of depth exponential with respect to the length of the formula. The existence of models can be checked by an alternating procedure which first guesses the type of the root, then guesses types of its children, and universally repeats the procedure for the children. This algorithm works in alternating polynomial space, and thus the problem is in ExpTime. A matching lower bound can be obtained as in [3].

[2] Please note that while the cited result does not consider reflexivity and symmetry, there are only some minor changes needed to cover these cases.

Table 1 Complexity of modal logics defined by consistent UHF^3 formulas.

A property implied by a formula	Global satisfiability	Local satisfiability
Polynomial model property	NP-c	NP-c
Semi-tree model property	ExpTime-c	PSpace-c
Transitive-tree model property	PSpace-c	PSpace-c

Transitive trees. This case can be treated similarly to the case of satisfiability over the class of transitive frames, i.e. the case of logic $\mathsf{K4}$ (see [9]). There are slight differences because in our case transitivity may fail at the last two elements of a path, however this detail does not cause real problems. We can also simply enforce infinite models (consider e.g. the class of irreflexive, transitive models and a modal formula $\top \wedge \Diamond\top \wedge \Box\Diamond\top$), so the length of paths cannot be bounded. However, we can bound polynomially the number of types on a path, which allows to show PSpace-completeness in both global and local cases.

Theorem 2 follows from the discussion above. The complexity results are summarized in Table 1.

References

1 R. Berger. The undecidability of the domino problem. *Mem. AMS*, 66, 1966.
2 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Comp. Sc.* Cambridge University Press, Cambridge, 2001.
3 Cheng-Chia Chen and I-Peng Lin. The complexity of propositional modal theories and the complexity of consistency of propositional modal theories. In *Logical Foundations of Computer Science*, volume 813 of *Lecture Notes in Computer Science*, pages 69–80. 1994.
4 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.
5 Yu. Sh. Gurevich and I. O. Koryakov. Remarks on berger's paper on the domino problem. *Siberian Mathematical Journal*, 13:319–321, 1972.
6 Edith Hemaspaandra. The price of universality. *Notre Dame Journal of Formal Logic*, 37(2):174–203, 1996.
7 Edith Hemaspaandra and Henning Schnoor. On the complexity of elementary modal logics. In Susanne Albers and Pascal Weil, editors, *STACS*, volume 1 of *LIPIcs*, pages 349–360. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.
8 Edith Hemaspaandra and Henning Schnoor. A universally defined undecidable unimodal logic. In Filip Murlak and Piotr Sankowski, editors, *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2011.
9 Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *Siam Journal on Computing*, 6:467–480, 1977.
10 Michael Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.
11 Martin Otto. Two variable first-order logic over ordered domains. *Journal of Symbolic Logic*, 66:685–702, 1998.
12 H. Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logic. *Proceedings of the Third Scandinavian Logic Symposium*, 1973.

The First-Order Theory of Ground Tree Rewrite Graphs

Stefan Göller[1] and Markus Lohrey*[2]

1 **Department of Computer Science, University of Bremen**
2 **Department of Computer Science, University of Leipzig**

Abstract

We prove that the complexity of the uniform first-order theory of ground tree rewrite graphs is in $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, O(n))$. Providing a matching lower bound, we show that there is a fixed ground tree rewrite graph whose first-order theory is hard for $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, \mathsf{poly}(n))$ with respect to logspace reductions. Finally, we prove that there is a fixed ground tree rewrite graph together with a single unary predicate in form of a regular tree language such that the resulting structure has a non-elementary first-order theory. For a long version of this paper with complete proofs see [11].

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases ground tree rewriting systems, first-order theories, complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.276

1 Introduction

Pushdown systems (PDS) are natural abstractions of sequential recursive programs. Moreover, the positive algorithmic properties of their transition graphs (pushdown graphs) make them attractive for the verification of sequential recursive programs. In particular, the model-checking problem for MSO (monadic second-order logic) and hence for most temporal logics (e.g. LTL, CTL) is decidable over pushdown graphs and precise complexity results are known (cf. [3, 14, 19, 20]). Ground tree rewrite systems [4, 8, 13] (GTRS), which are also known as ground term rewrite systems, generalize PDS from strings to trees. While the rules of a PDS rewrite a prefix of a given word, the rules of a GTRS rewrite a *subtree* of a given tree. GTRS can model (on an abstract level) concurrent programs with the ability to spawn new subthreads that are hierarchically structured, which in turn may terminate and return some values to their parents.

The transition graphs of GTRS (ground tree rewrite graphs) do not share all the nice algorithmic properties of pushdown graphs. For instance, the infinite grid is easily seen to be a ground tree rewrite graph, which implies that MSO is undecidable over GTRS. This holds even for most linear-time and branching-time temporal logics such as LTL and CTL (cf. [13, 17]). On the positive side, reachability, recurrent reachability, fair termination and certain fragments of LTL are decidable (cf. [4, 7, 13, 16, 17]). Moreover, first-order logic (FO) and first-order logic with reachability predicates are decidable [8]. This implies that model-checking of the CTL-fragment EF is decidable for GTRS; the precise complexity was recently clarified in [10].

* The second author is supported by the DFG project GELO.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 276–287

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While model-checking FO with reachability predicates is clearly non-elementary over GTRS (this holds already for the infinite binary tree, which is a pushdown graph [15]), the precise complexity of model-checking FO over GTRS is open, although the problem is known to be decidable since more than 20 years [8]. The algorithm provided in [8] has non-elementary complexity due to an exponential blowup when dealing with negation.

The main contribution of this paper solves this problem. We prove that (i) the first-order theory of every ground tree rewrite graph belongs to $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, O(n))$ (doubly exponential alternating time, where the number of alternations is bounded linearly) and (ii) that there exists a fixed ground tree rewrite graph with an $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, \mathsf{poly}(n))$-complete first-order theory. The upper bound of $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, O(n))$ even holds uniformly, which means that the GTRS may be part of the input. The complexity class $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, \mathsf{poly}(n))$ appears also in other contexts. For instance, Presburger Arithmetic (the first-order theory of $(\mathbb{N}, +)$) is known to be complete for $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, \mathsf{poly}(n))$ [1].

The upper bound of $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, \mathsf{poly}(n))$ is shown by the method of Ferrante and Rackoff [9]. Basically, the idea is to show the existence of a winning strategy of the duplicator in an Ehrenfeucht-Fraïssé game, where the duplicator chooses "small" elements. This method is one of the main tools for proving upper bounds for FO-theories. We divide the upper bound proof into two steps. In a first step, we reduce the FO-theory for a ground tree rewrite graph to the FO-theory for a very simple word rewrite graph, where all word rewrite rules replace one symbol by another symbol. The alphabet consists of all trees, whose size is bounded by a singly exponential function in the input size (hence, the alphabet size is doubly exponential in the input size; this is the reason for the doubly exponential time bound). Basically, we obtain a word over this alphabet from a tree t by cutting off some prefix-closed set C in the tree and taking the resulting sequence of trees. Intuitively, the set C consists of all nodes u of t such that the subtree rooted in u is "large". Here, "large" has to be replaced by a concrete value $m \in \mathbb{N}$ such that a sequence of n rewrite steps applied to a tree t cannot touch a node from the upward-closed set C. Clearly, m depends on n. In our context, n will be exponential in the input size and so will m. In a second step (see the long version [11]), we provide an upper bound for the FO-theory of a word rewrite graph of the above form.

For the lower bound, we show in this extended abstract only hardness for 2NEXP (doubly exponential nondeterministic time) using a $(2^{2^n} \times 2^{2^n})$ tiling problem (completeness for $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, \mathsf{poly}(n))$ is shown in the long version [10] using an alternating version of this tiling problem). In this problem, we are given a word w of length n over some fixed set of tiles, and it is asked, whether this word can be completed to a tiling of an array of size $(2^{2^n} \times 2^{2^n})$, where the word w is an initial part of the first row. There exists a fixed set of tiles, for which this problem is 2NEXP-complete. From this fixed set of tiles, we construct a fixed GTRS such that the following holds: From a given word w of length n over the tiles, one can construct (in logspace) a first-order formula that evaluates to true in our fixed ground tree rewrite graph if and only if the word w is a positive instance of the $(2^{2^n} \times 2^{2^n})$ tiling problem. Our construction is inspired by [10], where it is shown that the model-checking problem for a fragment of the logic EF (consisting of those EF-formulas, where on every path of the syntax tree at most one EF-operator occurs) over GTRS is $\mathsf{P}^{\mathsf{NEXP}}$-complete.

We finally state that there exists a fixed ground tree rewrite graph together with a single unary predicate in form of a regular tree language such that the resulting structure has a non-elementary first-order theory. This result is shown by a reduction from first-order satisfiability of finite binary words, which is non-elementary [15]. It should be noted that the first-order theory of a pushdown graph extended by regular unary predicates still has an elementary first-order theory: it is an automatic structure of bounded degree, hence its

first-order theory belongs to 2EXPSPACE [12]. However, we remark that ground tree rewrite graphs are not of bounded degree, hence the result from [12] for tree-automatic structures of bounded degree (stating that their first-order theories belong to 3EXPTIME) cannot be applied to obtain an elementary upper bound in our setting.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, \ldots\}$ be the set of *non-negative integers*. For $i, j \in \mathbb{N}$ we define the interval $[i, j] = \{i, i+1, \ldots, j\}$ and $[j] = [0, j]$. For an alphabet A (possibly infinite), we denote with $A^+ = A^* \setminus \{\varepsilon\}$ the set of all non-empty words over A. The length of the word $w \in A^*$ is denoted by $|w|$. For $B \subseteq A$, we denote with $|w|_B$ the number of occurrences of symbols from B in the word w. Let $f : A \to B$ be a mapping. For $A' \subseteq A$, we denote with $f{\restriction}A' : A' \to B$ the restriction of f to A'. For sets A, B, C (where A and B may have a non-empty intersection) and two mappings $f : A \to C$ and $g : B \to C$, we say that f and g are *compatible* if $f{\restriction}(A \cap B) = g{\restriction}(A \cap B)$. Finally, for mappings $f : A \to C$ and $g : B \to C$ with $A \cap B = \emptyset$, we define $f \uplus g : A \cup B \to C$ as the mapping with $(f \uplus g)(a) = f(a)$ for $a \in A$ and $(f \uplus g)(b) = g(b)$ for $b \in B$.

We will deal with alternating complexity classes, see [5] for more details. For functions $t(n)$ and $a(n)$ with $a(n) \leq t(n)$ for all $n \geq 0$ let $\mathsf{ATIME}(t(n), a(n))$ denote the class of all problems solvable on an alternating Turing-machine in time $t(n)$ with at most $a(n)$ alternations. We note that $\mathsf{ATIME}(t(n), t(n))$ is contained in $\mathsf{DSPACE}(t(n))$ if $t(n) \geq n$ [5].

2.1 Labelled graphs and first-order logic

A (directed) *graph* is a pair $(V, \to)$, where V is a set of *nodes* and $\to \subseteq V \times V$ is a binary relation. A *labelled graph* is a tuple $\mathfrak{G} = (V, \Sigma, \{\xrightarrow{a} \mid a \in \Sigma\})$, where V is a set of *nodes*, Σ is a finite set of *actions*, and $\xrightarrow{a}$ is a binary relation on V for all $a \in \Sigma$. We note that (labelled) graphs may have infinitely many nodes. We also write $v \in \mathfrak{G}$ for $v \in V$. For $u, v \in V$, we define $d_{\mathfrak{G}}(u, v)$ as the length of a shortest undirected path between u and v in the graph $(V, \bigcup_{a \in \Sigma} \xrightarrow{a})$. For $n \in \mathbb{N}$ and $u \in V$ let $S_n(\mathfrak{G}, u) = \{v \in V \mid d_{\mathfrak{G}}(u, v) \leq n\}$ be the *sphere* of radius n around u. Moreover, for $u_1, \ldots, u_k \in V$ let $S_n(\mathfrak{G}, u_1, \ldots, u_k) = \bigcup_{1 \leq i \leq k} S_n(\mathfrak{G}, u_i)$. We identify $S_n(\mathfrak{G}, u_1, \ldots, u_k)$ with the subgraph of $\mathfrak{G}$ induced by the set $S_n(\mathfrak{G}, u_1, \ldots, u_k)$, where in addition every u_i $(1 \leq i \leq k)$ is added as a constant. For two labelled graphs $\mathfrak{G}_1$ and $\mathfrak{G}_2$ and nodes $u_1, \ldots, u_k \in \mathfrak{G}_1$, $v_1, \ldots, v_k \in \mathfrak{G}_2$, we will consider isomorphisms $f : S_n(\mathfrak{G}_1, u_1, \ldots, u_k) \to S_n(\mathfrak{G}_2, v_1, \ldots, v_k)$. Such an isomorphism has to map u_i to v_i. We write $S_n(\mathfrak{G}_1, u_1, \ldots, u_k) \cong S_n(\mathfrak{G}_2, v_1, \ldots, v_k)$ if there is an isomorphism $f : S_n(\mathfrak{G}_1, u_1, \ldots, u_k) \to S_n(\mathfrak{G}_2, v_1, \ldots, v_k)$. The following lemma is straightforward.

▶ **Lemma 1.** *Let $\mathfrak{G}_1, \mathfrak{G}_2$ be labelled graphs with the same set of actions. Let $\overline{u} \in \mathfrak{G}_1^k$, $\overline{v} \in \mathfrak{G}_2^k$, $u \in \mathfrak{G}_1$, and $v \in \mathfrak{G}_2$ such that $u \notin S_{2n+1}(\mathfrak{G}_1, \overline{u})$ and $v \notin S_{2n+1}(\mathfrak{G}_1, \overline{v})$. Finally, let $f : S_n(\mathfrak{G}_1, \overline{u}) \to S_n(\mathfrak{G}_2, \overline{v})$ and $f' : S_n(\mathfrak{G}_1, u) \to S_n(\mathfrak{G}_2, v)$ be isomorphisms. Then $f \uplus f' : S_n(\mathfrak{G}_1, u, \overline{u}) \to S_n(\mathfrak{G}_2, v, \overline{v})$ is an isomorphism as well.*

Later, we have to lift a relation $\to$ from a set A to a larger set. We will denote this new relation again by $\to$. Two constructions will be needed. Assume that $\to$ is a binary relation on a set A and let $A \subseteq B$. We lift $\to$ to the set B^+ of non-empty words over B as follows: For all $u, v \in B^+$, we have $u \to v$ if and only if there are $x, y \in B^*$ and $a, b \in A$ such that $a \to b$ and $u = xay$, $v = xby$. Note that this implies $|u| = |v|$. The second construction lifts $\to \subseteq A \times A$ from A to $\mathbb{N} \times A$ as follows: For $a, b \in A$ and $m, n \in \mathbb{N}$ let $(m, a) \to (n, b)$ if and only if $m = n$ and $a \to b$. Note that $(\mathbb{N} \times A, \to)$ consists of $\aleph_0$ many disjoint copies of

$(A, \to)$. Finally, for a labelled graph $\mathfrak{G} = (V, \Sigma, \{\xrightarrow{a} | \ a \in \Sigma\})$, we define the labelled graph $\mathfrak{G}^+ = (V^+, \Sigma, \{\xrightarrow{a} | \ a \in \Sigma\})$. By the above definition, $\xrightarrow{a}$ is lifted to a relation on V^+.

We will consider first-order logic (FO) with equality over labelled graphs. Thus, for a set Σ of actions, we have for each $a \in \Sigma$ a binary relation symbol $a(x, y)$ in our language. The meaning of $a(x, y)$ is of course $x \xrightarrow{a} y$. If $\varphi(x_1, \ldots, x_n)$ is a first-order formula with free variables $x_1, \ldots, x_n$, $\mathfrak{G} = (V, \Sigma, \{\xrightarrow{a} | \ a \in \Sigma\})$ is a labelled graph, and $v_1, \ldots, v_n \in V$, then we write $\mathfrak{G} \models \varphi(v_1, \ldots, v_n)$ if φ evaluates to true in $\mathfrak{G}$, when variable x_i is instantiated by v_i $(1 \leq i \leq n)$. The *first-order theory* of a labelled transition graph $\mathfrak{G}$ is the set of all first-order sentences (i.e., first-order formulas without free variables) φ with $\mathfrak{G} \models \varphi$. The *quantifier rank* of a first-order formula is the maximal number of nested quantifiers in φ.

2.2 Trees and ground tree rewrite systems

Let $\preceq$ denote the prefix order on $\mathbb{N}^*$, i.e., $x \preceq y$ for $x, y \in \mathbb{N}^*$ if there exists $z \in \mathbb{N}^*$ with $y = xz$. A set $D \subseteq \mathbb{N}^*$ is called *prefix-closed* if for all $x, y \in \mathbb{N}^*$, $x \preceq y \in D$ implies $x \in D$. A *ranked alphabet* is a collection of finite and pairwise disjoint alphabets $A = (A_i)_{i \in [k]}$ for some $k \geq 0$ such that $A_0 \neq \emptyset$. For simplicity we identify A with $\bigcup_{i \in [k]} A_i$. A *ranked tree* over the ranked alphabet A is a mapping $t : D_t \to A$, where (i) D_t is non-empty, finite, and prefix-closed, and (ii) for each $x \in D_t$ with $t(x) \in A_i$ we have $x1, \ldots, xi \in D_t$ and $xj \notin D_t$ for each $j > i$. By Tr_A we denote the set of all ranked trees over the ranked alphabet A. Let $t \in \mathsf{Tr}_A$. Elements of D_t are called *nodes*. A *leaf* of t is a node x with $t(x) \in A_0$. An *internal node* of t is a node, which is not a leaf. We also refer to $\varepsilon \in D_t$ as the *root* of t. Let $\mathsf{size}(t) = |D_t|$ be the size of the tree t. It is easy to show that $|\{t \in \mathsf{Tr}_A \mid \mathsf{size}(t) \leq n\}| \leq |A|^n$. For $x \in [1, k]^*$ we define $xD_t = \{xy \in [1, k]^* \mid y \in D_t\}$ and $x^{-1}D_t = \{y \in [1, k]^* \mid xy \in D_t\}$. By $t^{\downarrow x}$ we denote the *subtree of* t with root x; it is defined by $D_{t^{\downarrow x}} = x^{-1}D_t$ and $t^{\downarrow x}(y) = t(xy)$. For a second tree $s \in \mathsf{Tr}_A$ and $x \in D_t$ we denote by $t[x/s]$ the tree that is obtained by replacing $t^{\downarrow x}$ in t by s. More formally, $D_{t[x/s]} = (D_t \setminus xD_{t^{\downarrow x}}) \cup xD_s$, $t[x/s](y) = t(y)$ for $y \in D_t \setminus xD_{t^{\downarrow x}}$, and $t[x/s](xz) = s(z)$ for $z \in D_s$.

Let C be a prefix-closed subset of D_t. We define the string of subtrees $t \setminus C$ as follows: If $C = \emptyset$, then $t \setminus C = t$. If $C \neq \emptyset$, then $t \setminus C = t^{\downarrow v_1} \cdots t^{\downarrow v_m}$, where $v_1, \ldots, v_m$ is a list of all nodes from $((C \cdot \mathbb{N}) \cap D_t) \setminus C$ in lexicographic order. Intuitively, we remove from the tree t the prefix-closed subset C and list all remaining maximal subtrees. For $n \in \mathbb{N}$ and a tree t we define the prefix-closed subset $\mathsf{up}(t, n) \subseteq D_t$ as $\mathsf{up}(t, n) = \{v \in D_t \mid \mathsf{size}(t^{\downarrow v}) > n\}$. Note that $t \setminus \mathsf{up}(t, n)$ is a list of all maximal subtrees of size at most n in t.

A *ground tree rewrite system (GTRS)* is tuple $\mathcal{R} = (A, \Sigma, R)$, where A is a ranked alphabet, Σ is finite set of actions, and $R \subseteq \mathsf{Tr}_A \times \Sigma \times \mathsf{Tr}_A$ is a finite set of rewrite rules. A rule (s, a, t) is also written as $s \xmapsto{a} t$. The corresponding *ground tree rewrite graph* is $\mathfrak{G}(\mathcal{R}) = (\mathsf{Tr}_A, \Sigma, \{\xrightarrow{a} | \ a \in \Sigma\})$, where for each $a \in \Sigma$, we have $t \xrightarrow{a} t'$ if and only if there exists a rule $(s \xmapsto{a} s') \in R$ and $x \in D_t$ such that $t^{\downarrow x} = s$ and $t' = t[x/s']$. The following two lemmas are not very hard to prove.

▶ **Lemma 2.** *Let $\mathcal{R} = (A, \Sigma, R)$ be a GTRS and let r be the maximal size of a tree that appears in R. Let s and t be ranked trees such that $d_{\mathfrak{G}(\mathcal{R})}(s, t) \leq n$. Then $\mathsf{size}(t) \leq \mathsf{size}(s) + r \cdot n$.*

▶ **Lemma 3.** *Let $\mathcal{R} = (A, \Sigma, R)$ be a GTRS and let r be the maximal size of a tree that appears in R. Let t be a ranked tree, $k \in \mathbb{N}$, and let $C \subseteq \mathsf{up}(t, r \cdot k)$ be prefix-closed. Then we have $S_k(\mathfrak{G}(\mathcal{R}), t) \cong S_k(\mathfrak{G}(\mathcal{R})^+, t \setminus C)$ (where $\mathfrak{G}(\mathcal{R})^+$ is defined in Section 2.1).*

3 An $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, O(n))$ upper bound

In this section we will prove the following result:

▶ **Theorem 4.** *The problem of checking $\mathfrak{G}(\mathcal{R}) \models \varphi$ for a given GTRS $\mathcal{R} = (A, \Sigma, R)$ and an FO-sentence φ over the signature of $\mathfrak{G}(\mathcal{R})$ belongs to the class $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, O(n))$.*

It suffices to prove Thm. 4 for the case that the ranked alphabet A contains a symbol of rank at least two. A ground tree rewrite graph, where all symbols have rank at most 1 is in fact a suffix rewrite graph, i.e., pushdown graph. But every pushdown graph is first-order interpretable in a full $|\Gamma|$-ary tree Γ^* (with Γ finite), where the defining first-order formulas can be easily computed from the pushdown automaton. Finally, the first-order theory of a full tree Γ^* (with $|\Gamma| \geq 2$) is complete for the class $\mathsf{ATIME}(2^{O(n)}, O(n))$ [6, 18].

The proof of Thm. 4 will be divided into two steps. In a first step, we reduce the FO-theory for a given ground tree rewrite graph to the FO-theory for a very simple word rewrite graph of the form $\mathfrak{G}^+$, where $\mathfrak{G}$ is a finite labelled graph. Note that if V is the set of nodes of $\mathfrak{G}$, then V^+ is the set of nodes of $\mathfrak{G}^+$. Moreover, every edge in $\mathfrak{G}^+$ replaces a single symbol in a word by another symbol. In our reduction, the size of the set V is doubly exponential in the input size (which is the size of the input formula plus the size of the input GTRS). In a second step, we solve the FO-theory of a simple word structure $\mathfrak{G}^+$ on an alternating Turing machine. More precisely, in the long version [11] of this paper, we prove:

▶ **Theorem 5.** *There exists an alternating Turing-machine M, which accepts all pairs $(\mathfrak{G}, \varphi)$, where $\mathfrak{G}$ is a finite labelled graph and φ is an FO-sentence over the signature of $\mathfrak{G}$ with $\mathfrak{G}^+ \models \varphi$. Moreover, M runs in time $O(n^\ell \cdot |\varphi|)$, where n is the number of nodes of $\mathfrak{G}$ and ℓ is the quantifier rank of φ. Moreover, the number of alternations is bounded by $O(\ell)$.*

The proof of Thm. 5 uses an application of the method of Ferrante and Rackoff [9], which is one of most successful techniques for proving upper bounds for the complexity of first-order theories. In the rest of this section, we will derive Thm. 4 from Thm. 5.

Fix a GTRS $\mathcal{R} = (A, \Sigma, R)$. We restrict to the case $A_1 \neq \emptyset \neq A_2$; the general case (see [11]) is technically more complicated, but the main idea is the same. Let $\mathfrak{G} = \mathfrak{G}(\mathcal{R})$ and let

$$\varphi = Q_\ell x_\ell \cdots Q_1 x_1 \; Q_0 x_0 : \psi$$

be a FO-sentence of quantifier rank $\ell + 1$, where $Q_0, \ldots, Q_\ell \in \{\forall, \exists\}$ and ψ is quantifier-free. We want to check, whether $\mathfrak{G} \models \varphi$. Let r be the maximal size of a tree that appears in R, and let $p \geq 2$ the maximal rank of a symbol from A. Let us define the following subsets of Tr_A (where $0 \leq i \leq \ell$):

$$\begin{aligned} U &= \{t \in \mathsf{Tr}_A \mid \mathsf{size}(t) \leq r \cdot (p+1) \cdot 4^\ell + 1\}, \\ V_i &= \{t \in \mathsf{Tr}_A \mid \mathsf{size}(t) \leq r \cdot 4^i\} \subseteq U \text{ and} \\ W_i &= \{\alpha(u_1, \ldots, u_q) \mid q \geq 1, \alpha \in A_q, u_1, \ldots, u_q \in V_i\} \setminus V_i \subseteq U. \end{aligned}$$

Intuitions on these sets will be given below. Note that $\mathsf{size}(t) \leq r \cdot p \cdot 4^i + 1$ for all $t \in W_i$. We consider the set U as a finite alphabet and the sets V_i and W_i as subalphabets. Note that $|U| \leq |A|^{r \cdot (p+1) \cdot 4^\ell + 1}$. On the set $(\mathbb{N} \times U^+) \cup U$ we define a labelled graph with the set of actions Σ. Take an action $\sigma \in \Sigma$. By our lifting constructions from Section 2.1, the binary relation $\xrightarrow{\sigma}$ on Tr_A is implicitly lifted to a binary relation on Tr_A^+ and $\mathbb{N} \times \mathsf{Tr}_A^+$. Since $(\mathbb{N} \times U^+) \cap U = \emptyset$, $\xrightarrow{\sigma}$ can be viewed as a binary relation on $(\mathbb{N} \times U^+) \cup U$; simply take the disjoint union of the relations on $(\mathbb{N} \times U^+)$ and U. We define the labelled graph

$$\mathfrak{S}_1 = ((\mathbb{N} \times U^+) \cup U, \; \Sigma, \; \{\xrightarrow{\sigma} \mid \sigma \in \Sigma\}).$$

For $0 \leq i \leq \ell$ and nodes $s_{i+1}, \ldots, s_\ell \in (\mathbb{N} \times U^+) \cup U$ define the following nodes sets in $\mathfrak{S}_1$:

$$L_i = (\mathbb{N} \times V_i^* W_i V_i^*) \cup V_i, \quad L_i(s_{i+1}, \ldots, s_\ell) = L_i \cup S_{3 \cdot 4^i}(\mathfrak{S}_1, s_{i+1}, \ldots, s_\ell).$$

Finally, define the FO-sentence (with relativized quantifiers)

$$\varphi_1 \;=\; Q_\ell x_\ell \in L_\ell \; Q_{\ell-1} x_{\ell-1} \in L_{\ell-1}(x_\ell) \;\cdots\; Q_0 x_0 \in L_0(x_1, \ldots, x_\ell) : \psi \tag{1}$$

over the signature of $\mathfrak{S}_1$ Note that in φ_1 the constraint set for a variable x_i depends on the values for the already quantified variables $x_{i+1}, \ldots, x_\ell$. Based on the following lemma, we show that $\mathfrak{G} \models \varphi$ if and only if $\mathfrak{S}_1 \models \varphi_1$.

▶ **Lemma 6.** *Assume that* $0 \leq i \leq \ell$,

- $\overline{s} = (s_{i+1}, \ldots, s_\ell) \in ((\mathbb{N} \times U^+) \cup U)^{\ell - i}$ *with* $s_j \in L_j \cup S_{3 \cdot 4^j}(\mathfrak{S}_1, s_{j+1}, \ldots, s_\ell)$ *for all* $j \in [i+1, \ell]$,
- $\overline{t} = (t_{i+1}, \ldots, t_\ell) \in \mathsf{Tr}_A^{\ell - i}$, *and*
- $f : S_{4^{i+1}}(\mathfrak{S}_1, \overline{s}) \to S_{4^{i+1}}(\mathfrak{G}, \overline{t})$ *is an isomorphism such that* $f \restriction S_{4^{i+1}}(\mathfrak{S}_1, s_j)$ *is the identity for all* $j \in [i+1, \ell]$ *with* $t_j \in V_{i+1}$ *or* $s_j \in V_{i+1}$.[1]

Then, the following holds:

(a) For all $t_i \in \mathsf{Tr}_A$ *there exists* $s_i \in L_i \cup S_{3 \cdot 4^i}(\mathfrak{S}_1, \overline{s})$ *and an isomorphism* $g : S_{4^i}(\mathfrak{S}_1, s_i, \overline{s}) \to S_{4^i}(\mathfrak{G}, t_i, \overline{t})$ *such that* f *and* g *are compatible and* $g \restriction S_{4^i}(\mathfrak{S}_1, s_j)$ *is the identity for all* $j \in [i, \ell]$ *with* $t_j \in V_i$ *or* $s_j \in V_i$.

(b) For all $s_i \in L_i \cup S_{3 \cdot 4^i}(\mathfrak{S}_1, \overline{s})$ *there exists* $t_i \in \mathsf{Tr}_A$ *and an isomorphism* $g : S_{4^i}(\mathfrak{S}_1, s_i, \overline{s}) \to S_{4^i}(\mathfrak{G}, t_i, \overline{t})$ *such that* f *and* g *are compatible and* $g \restriction S_{4^i}(\mathfrak{S}_1, s_j)$ *is the identity for all* $j \in [i, \ell]$ *with* $t_j \in V_i$ *or* $s_j \in V_i$.

Before we prove the lemma, let us provide some intuition. For case (a) we will basically distinguish two cases: In case t_i is "close" to some tree in the tuple $\overline{t}$, then the simulating s_i can safely be chosen as t_i itself. In case t_i is "far" to all trees in $\overline{t}$, we distinguish two cases: Either the size of t_i exceeds $r \cdot 4^i$ or not. If it does, then s_i will be chosen as a pair from $\{n\} \times V_i^* W_i V_i^*$ for some fresh number n that does not appear as a first component of any element in $\overline{t}$, and where the second component of s_i consists basically of $t_i \setminus C$ for some prefix-closed subset C of t_i's nodes. Intuitively, this means that s_i does not have to be "too big" in order to simulate t_i: only "small" subtrees of t_i have to be accounted for. Lemma 3 will be crucial. In case $|t_i| \leq r \cdot 4^i$, we can prove that we can set $s_i = t_i \in V_i$. For case (b) we can proceed similarly, but the main crux is that for each element $s_i \in \mathbb{N} \times V_i^* W_i V_i^*$ we can build a tree $t_i \in \mathsf{Tr}_A$ such that the spheres of radius 4^i around s_i and t_i are isomorphic.

PROOF (of Lemma 6). Let $f : S_{4^{i+1}}(\mathfrak{S}_1, \overline{s}) \to S_{4^{i+1}}(\mathfrak{G}, \overline{t})$ be an isomorphism such that $f \restriction S_{4^{i+1}}(\mathfrak{S}_1, s_j)$ is the identity for all $i+1 \leq j \leq \ell$ with $t_j \in V_{i+1}$ or $s_j \in V_{i+1}$. Let us first prove statement (a). Let $t_i \in \mathsf{Tr}_A$. We distinguish two cases:

Case 1. $t_i \in S_{3 \cdot 4^i}(\mathfrak{G}, \overline{t})$. Thus, t_i belongs to the range of the isomorphism f. Moreover, $S_{4^i}(\mathfrak{G}, t_i, \overline{t}) \subseteq S_{4^{i+1}}(\mathfrak{G}, \overline{t})$. Then, we set $s_i = f^{-1}(t_i) \in S_{3 \cdot 4^i}(\mathfrak{S}_1, \overline{s})$. We define g as the restriction of f to the set $S_{4^i}(\mathfrak{S}_1, s_i, \overline{s}) \subseteq S_{4^{i+1}}(\mathfrak{S}_1, \overline{s})$. Now, assume that $t_i \in V_i$, i.e., $\mathsf{size}(t_i) \leq r \cdot 4^i$. We have to show that $g \restriction S_{4^i}(\mathfrak{S}_1, s_i)$ is the identity. Let t_j $(j > i)$ be such that $d_{\mathfrak{G}}(t_i, t_j) \leq 3 \cdot 4^i$. Lemma 2 implies $\mathsf{size}(t_j) \leq \mathsf{size}(t_i) + r \cdot 3 \cdot 4^i \leq r \cdot 4^i + r \cdot 3 \cdot 4^i = r \cdot 4^{i+1}$. Hence, $t_j \in V_{i+1}$ and $f \restriction S_{4^{i+1}}(\mathfrak{S}_1, s_j)$ is the identity by assumption. Since $S_{4^i}(\mathfrak{S}_1, s_i) \subseteq S_{4^{i+1}}(\mathfrak{S}_1, s_j)$, it follows that $g \restriction S_{4^i}(\mathfrak{S}_1, s_i)$ is the identity too. If $s_i \in V_i$, then we can argue analogously.

[1] For $i = \ell$, f is the isomorphism between empty sets.

Case 2. $t_i \notin S_{3\cdot 4^i}(\mathfrak{G}, \bar{t})$. Thus, $t_i \notin S_{2\cdot 4^i+1}(\mathfrak{G}, \bar{t})$. We will find $s_i \in L_i$ and an isomorphism $f' : S_{4^i}(\mathfrak{S}_1, s_i) \to S_{4^i}(\mathfrak{G}, t_i)$ such that $s_i \notin S_{3\cdot 4^i}(\mathfrak{S}_1, \bar{s})$. Then, Lemma 1 implies that $g = (f \restriction S_{4^i}(\mathfrak{S}_1, \bar{s})) \uplus f'$ is an isomorphism from $S_{4^i}(\mathfrak{S}_1, s_i, \bar{s})$ to $S_{4^i}(\mathfrak{G}, t_i, \bar{t})$, which is compatible with f. Moreover, we will show that if $t_i \in V_i$ or $s_i \in V_i$, then f' is the identity. In order to define s_i, let $t_i \setminus \mathsf{up}(t_i, r \cdot 4^i) = u_1 \cdots u_m$. Thus $u_1, \ldots, u_m \in V_i$. Choose a number $n \in \mathbb{N}$ such that n does not appear as a first component of a pair from $\{s_{i+1}, \ldots, s_\ell\} \cap (\mathbb{N} \times U^+)$.

Case 2.1. $\mathsf{size}(t_i) > r \cdot 4^i$. Then there exists a symbol $\alpha \in A$ of rank $q \geq 1$, an index $1 \leq j \leq m-q+1$, and a prefix-closed subset $C \subseteq \mathsf{up}(t_i, r\cdot 4^i)$ such that $\alpha(u_j, \ldots, u_{j+q-1}) \in W_i$ and $t_i \setminus C = u_1 u_2 \cdots u_{j-1} \alpha(u_j, \ldots, u_{j+q-1}) u_{j+q} \cdots u_m$. We have $t_i \setminus C \in V_i^* W_i V_i^*$. Let $s_i = (n, t_i \setminus C) \in L_i$. Due to the choice of n, we have $s_i \notin S_\rho(\mathfrak{S}_1, \bar{s})$ for all ρ. Moreover, Lemma 2 and the definition of the set U implies that the sphere of radius 4^i around every tree from $u_1, u_2, \ldots, u_{j-1}$, $\alpha(u_j, \ldots, u_{j+q-1})$, $u_{j+q}, \ldots, u_m$ is completely contained in U. With Lemma 3 (setting $k = 4^i$), we get $S_{4^i}(\mathfrak{S}_1, s_i) \cong S_{4^i}(\mathfrak{G}, t_i)$ via an isomorphism f'. Finally, $\mathsf{size}(t_i) > r \cdot 4^i$ and $s_i \notin U$. Hence, neither $s_i \in V_i$ nor $t_i \in V_i$.

Case 2.2. $\mathsf{size}(t_i) \leq r \cdot 4^i$. Hence, $m = 1$ and $t_i = u_1$. We set $s_i = t_i = u_1 \in V_i \subseteq L_i$. Thus $\mathsf{size}(s_i) = \mathsf{size}(t_i) \leq r \cdot 4^i$. We have $S_{4^i}(\mathfrak{G}, t_i) \subseteq U$, which implies $S_{4^i}(\mathfrak{S}_1, s_i) = S_{4^i}(\mathfrak{G}, t_i)$. Assume that $s_i \in S_{3\cdot 4^i}(\mathfrak{S}_1, \bar{s})$. We will deduce a contradiction. Let $j > i$ such that $d_{\mathfrak{S}_1}(s_i, s_j) \leq 3 \cdot 4^i$. Since $s_i \in U$, we must have $s_j \in U$ as well (there is no path in $\mathfrak{S}_1$ between the sets U and $\mathbb{N} \times U^+$). Moreover, Lemma 2 implies $\mathsf{size}(s_j) \leq \mathsf{size}(s_i) + r \cdot 3 \cdot 4^i \leq r \cdot 4^{i+1}$, i.e., $s_j \in V_{i+1}$. Hence, $f \restriction S_{4^{i+1}}(\mathfrak{S}_1, s_j)$ is the identity and $t_i \in S_{3\cdot 4^i}(\mathfrak{G}, t_j)$, a contradiction. We can finally choose for f' the identity isomorphism on $S_{4^i}(\mathfrak{S}_1, s_i) = S_{4^i}(\mathfrak{G}, t_i)$. This proves (a).

Let us now prove (b). Let $s_i \in L_i \cup S_{3\cdot 4^i}(\mathfrak{S}_1, \bar{s})$. Again, we distinguish two cases.

Case 1. $s_i \in S_{3\cdot 4^i}(\mathfrak{S}_1, \bar{s})$. This implies $S_{4^i}(\mathfrak{S}_1, s_i, \bar{s}) \subseteq S_{4^{i+1}}(\mathfrak{S}_1, \bar{s})$. We set $t_i = f(s_i) \in S_{3\cdot 4^i}(\mathfrak{G}, \bar{t})$. We can conclude as in Case 1 for point (a) above.

Case 2. $s_i \notin S_{3\cdot 4^i}(\mathfrak{S}_1, \bar{s})$. Hence, $s_i \in L_i$. We will find $t_i \in \mathsf{Tr}_A$ and an isomorphism $f' : S_{4^i}(\mathfrak{S}_1, s_i) \to S_{4^i}(\mathfrak{G}, t_i)$ such that $t_i \notin S_{3\cdot 4^i}(\mathfrak{G}, \bar{t})$. Then, Lemma 1 implies that the mapping $g = (f \restriction S_{4^i}(\mathfrak{S}_1, \bar{s})) \uplus f'$ is an isomorphism from $S_{4^i}(\mathfrak{S}_1, s_i, \bar{s})$ to $S_{4^i}(\mathfrak{G}, t_i, \bar{t})$, which is compatible with f. Moreover, we will show that if $t_i \in V_i$ or $s_i \in V_i$, then f' is the identity.

Case 2.1. $s_i \in V_i \subseteq \mathsf{Tr}_A$. We set $t_i = s_i$. Hence, $\mathsf{size}(t_i) \leq r \cdot 4^i$ and one can argue analogously to Case 2.2 in the proof for statement (a).

Case 2.2. $s_i \in \mathbb{N} \times V_i^* W_i V_i^*$. Let $s_i = (n, w)$ where $w \in V_i^* W_i V_i^*$. Hence, $w = u_1 \cdots u_m$ with $u_1, \ldots, u_m \in V_i \cup W_i$. Moreover, there is a unique index j such that $u_j \in W_i$. Assume that $u_j = \alpha(u'_1, \ldots, u'_q)$ with $\alpha \in A$ and $q \geq 1$. By the definition of the set W_i we have $u'_1, \ldots, u'_q \in V_i$. Since we assume that $A_1 \neq \emptyset \neq A_2$, we can choose for t_i a tree with the following properties:

- $t_i \setminus \mathsf{up}(t_i, r \cdot 4^i) = u_1 \cdots u_{j-1} u'_1 \cdots u'_q u_{j+1} \cdots u_m$. For this, we connect all trees $u_1, \ldots, u_m$ to one tree using a chain of binary symbols, starting from $u_j \in W_i$.
- $t_i \notin S_{3\cdot 4^i}(\mathfrak{G}, \bar{t})$. This can be enforced by adding a long enough chain of unary symbols to the root.

With Lemma 3, the first point implies $S_{4^i}(\mathfrak{S}_1, s_i) \cong S_{4^i}(\mathfrak{G}, t_i)$. Moreover, since t_i contains a subtree from W_i, we have $t_i \notin V_i$. ◀

Using the classical back-and-forth argument from the proof of the Ehrenfeucht-Fraïssé-Theorem, we can deduce the following lemma from Lemma 6.

▶ **Lemma 7.** *Assume that* $-1 \leq i \leq \ell$,

- $\overline{s} = (s_{i+1}, \ldots, s_\ell) \in ((\mathbb{N} \times U^+) \cup U)^{\ell-i}$ *with* $s_j \in L_j \cup S_{3\cdot 4^j}(s_{j+1}, \ldots, s_\ell)$ *for* $j \in [i+1, \ell]$,
- $\overline{t} = (t_{i+1}, \ldots, t_\ell) \in \mathsf{Tr}_A^{\ell-i}$, *and*
- $f : S_{4^{i+1}}(\mathfrak{S}_1, \overline{s}) \to S_{4^{i+1}}(\mathfrak{G}, \overline{t})$ *is an isomorphism such that* $f \restriction S_{4^{i+1}}(\mathfrak{S}_1, s_j)$ *is the identity for all* $j \in [i+1, \ell]$ *with* $t_j \in V_{i+1}$ *or* $s_j \in V_{i+1}$.

Then, for every quantifier-free FO-formula ψ *over the signature of* $\mathfrak{G}$ *and all* $Q_0, \ldots, Q_i \in \{\forall, \exists\}$ *we have:* $\mathfrak{S}_1 \models Q_i x_i \in L_i(\overline{s}) \cdots Q_0 x_0 \in L_0(x_1, \ldots, x_i, \overline{s}) : \psi(x_0, \ldots, x_i, \overline{s})$ *if and only if* $\mathfrak{G} \models Q_i x_i \cdots Q_0 x_0 : \psi(x_0, \ldots, x_i, \overline{t})$.

Setting $i = \ell$ in Lemma 7, it follows $\mathfrak{G} \models \varphi$ if and only if $\mathfrak{S}_1 \models \varphi_1$, where φ_1 is from (1). It now requires rather easy but technical arguments to compute a finite labelled graph $\mathfrak{G}'$ with doubly exponentially (in our input size) many nodes and a first-order sentence φ' (of polynomial size and quantifier rank $O(\ell)$) such that $\mathfrak{S}_1 \models \varphi_1$ if and only if $(\mathfrak{G}')^+ \models \varphi'$. To get rid of the direct product with $\mathbb{N}$ in the node set of $\mathfrak{S}_1$, we use the simple fact that for an arbitrary graph $(V, \to)$, $((V \cup \{\$\})^+ \setminus \{\$\}^+, \to)$ (where $\$ \notin V$ is a new symbol) is isomorphic to $(\mathbb{N} \times V^+, \to)$; here we use the lifting constructions from Section 2.1. Then, Thm. 4 can be easily derived from Thm. 5. We refer to [11] for details.

4 A lower bound

In the long version of this paper [11], we prove the following lower bound:

▶ **Theorem 8.** *There is a fixed GTRS* $\mathcal{R}$ *such that the first-order theory of* $\mathfrak{G}(\mathcal{R})$ *is hard for* $\mathsf{ATIME}(2^{2^{\mathsf{poly}(n)}}, \mathsf{poly}(n))$ *under logspace reductions.*

In this section, we will sketch a proof for the slightly weaker lower bound of 2NEXP. This will be achieved using a tiling problem. Tiling problems turned out to be an important tool for proving lower bounds in logic, see e.g. [2]. So, let us start with a few definitions concerning tiling systems. A *tiling system* is a tuple $S = (\Theta, \mathbb{H}, \mathbb{V})$, where Θ is a finite set of *tile types*, $\mathbb{H} \subseteq \Theta \times \Theta$ is a *horizontal matching relation*, and $\mathbb{V} \subseteq \Theta \times \Theta$ is a *vertical matching relation*. A mapping $\sigma : [0, k-1] \times [0, k-1] \to \Theta$ (where $k \geq 0$) is a *k-solution for S* if for all $x, y \in [0, k-1]$ the following holds: (i) if $x < k-1$, $\sigma(x, y) = \theta$, and $\sigma(x+1, y) = \theta'$, then $(\theta, \theta') \in \mathbb{H}$, and (ii) if $y < k-1$, $\sigma(x, y) = \theta$, and $\sigma(x, y+1) = \theta'$, then $(\theta, \theta') \in \mathbb{V}$. Let $\mathsf{Sol}_k(S)$ denote the set of all k-solutions for S. Let $w = \theta_0 \cdots \theta_{n-1} \in \Theta^n$ be a word and let $k \geq n$. With $\mathsf{Sol}_k(S, w)$ we denote the set of all $\sigma \in \mathsf{Sol}_k(S)$ such that $\sigma(x, 0) = \theta_x$ for all $x \in [0, n-1]$. For a fixed tiling system S, its $(2^{2^n} \times 2^{2^n})$ *tiling problem* asks for a given word $w \in \Theta^n$, whether $\mathsf{Sol}_{2^{2^n}}(S, w) \neq \emptyset$ holds. Using the standard encoding of Turing machine computations by tilings, it follows easily that there exists a fixed tiling system S_0 whose $(2^{2^n} \times 2^{2^n})$ tiling problem is 2NEXP-hard under logspace reductions; see also [2]. Let us fix such a tiling system $S_0 = (\Theta_0, \mathbb{H}_0, \mathbb{V}_0)$ for the rest of the section.

We now define a fixed GTRS $\mathcal{R}_0 = (A, \Sigma, R)$ and prove that the first-order theory of $\mathfrak{G}(\mathcal{R}_0)$ is 2NEXP-hard under logspace reductions. We define $A_0 = \{\heartsuit, \not\Vdash, \not\Vdash_\dagger, \not\Vdash_\ddagger, \mathbb{O}, \mathbb{O}_\dagger, \mathbb{O}_\ddagger\}$, $A_1 = \Theta_0$, $A_2 = \{\bullet\}$, and $\Sigma = \{\ell, r, h, u, m_\dagger, m_\ddagger, \} \cup \Theta_0 \cup A_0$. The set of rewrite rules R is given as follows:

$$X \overset{X}{\longmapsto} X \text{ for all } X \in A_0 \qquad \theta(X_\ddagger) \overset{\theta}{\longmapsto} \theta(X_\ddagger) \text{ for all } \theta \in \Theta_0, X \in \{\not\Vdash, \mathbb{O}\}$$

$$X \overset{m_\dagger}{\longmapsto} X_\dagger \text{ for all } X \in \{\not\Vdash, \mathbb{O}\} \qquad \bullet(\heartsuit, \heartsuit) \overset{u}{\longmapsto} \heartsuit$$

$$X_\dagger \overset{m_\ddagger}{\longmapsto} X_\ddagger \text{ for all } X \in \{\not\Vdash, \mathbb{O}\} \qquad \bullet(\heartsuit, X_\ddagger) \overset{r}{\longmapsto} X_\ddagger \text{ for all } X \in \{\not\Vdash, \mathbb{O}\}$$

$$X_\dagger \overset{h}{\longmapsto} \heartsuit \text{ for all } X \in \{\not\Vdash, \mathbb{O}\} \qquad \bullet(X_\ddagger, \heartsuit) \overset{\ell}{\longmapsto} X_\ddagger \text{ for all } X \in \{\not\Vdash, \mathbb{O}\}$$

For the rest of this section we fix $\mathfrak{G}_0 = \mathfrak{G}(\mathcal{R}_0)$ and an input $w = \theta_0 \cdots \theta_{n-1} \in \Theta_0^n$ of the $(2^{2^n} \times 2^{2^n})$ tiling problem for S_0. Our goal is to compute in logspace from w a first-order sentence φ over Σ such that $\mathsf{Sol}_{2^{2^n}}(S_0, w) \neq \emptyset$ if and only if $\mathfrak{G}_0 \models \varphi$. We will need the following lemma, which goes back to the work of Fischer and Rabin.

▶ **Lemma 9.** *Given a subset of actions $\Gamma \subseteq \Sigma$ and the binary representation of $j \in [0, 2^{n+1}]$, one can compute in logspace a first-order formula $\Gamma^j(x, y)$ such that for all $t, t' \in \mathsf{Tr}_A$ we have $\mathfrak{G}_0 \models \Gamma^j(t, t')$ if and only if there is a path of length j from t to t' in the graph $(\mathsf{Tr}_A, \bigcup_{\gamma\in\Gamma} \xrightarrow{\gamma})$.*

If $\Gamma = \{\gamma\}$, we write $\gamma^j(x, y)$ for the formula $\Gamma^j(x, y)$. For $\Gamma_1, \ldots, \Gamma_k \subseteq \Sigma$ and $j_1, \ldots, j_k \in \mathbb{N}$, we write $[\Gamma_1^{j_1} \cdots \Gamma_k^{j_k}](x, y)$ for $\exists x_0, \ldots, x_k : \big(x_0 = x \wedge x_k = y \wedge \bigwedge_{i=1}^{k} \Gamma_i^{j_i}(x_{i-1}, x_i)\big)$.

A tree $t \in \mathsf{Tr}_A$ is a *tile tree* if $t = \theta(t')$ for some $\theta \in \Theta_0$ and $t' \in \mathsf{Tr}_{\{\mathbb{O},\mathbb{1},\bullet\}}$ such that $\{1, 2\}^{n+1}$ is the set of leaves of t'. Fix a tile tree $t = \theta(t')$. Then t has precisely $2^{n+1} = 2 \cdot 2^n$ leaves. For a leaf λ of t let $\mathsf{lex}(\lambda) \in [0, 2^{n+1} - 1]$ be the position of λ among all leaves w.r.t. the lexicographic order (starting with 0). The intention is that t represents the θ-labeled grid element $(M, N) \in [0, 2^{2^n} - 1]^2$, where each leaf λ that is a left (resp. right) child represents the $\lfloor \frac{\mathsf{lex}(\lambda)}{2} \rfloor^{\text{th}}$ least significant bit of the 2^n-bit binary presentation of M (resp. N): In case λ is a left child, then $t(\lambda) = \mathbb{O}$ (resp. $t(\lambda) = \mathbb{1}$) if and only if the $\lfloor \frac{\mathsf{lex}(\lambda)}{2} \rfloor^{\text{th}}$ least significant bit of M equals 0 (resp. 1) and analogously if λ is a right child this corresponds to N. We say a leaf λ of a tree t is *marked* (resp. *selected*) if $t(\lambda) = X_\dagger$ (resp. $t(\lambda) = X_\ddagger$) for some $X \in \{\mathbb{O}, \mathbb{1}\}$. A *marked tile tree* is a tree that can be obtained from a tile tree t by marking *every* leaf of t. For the rest of this section, let $D = 2^{n+1} - (n + 2)$.

▶ **Lemma 10.** *One can compute in logspace a first-order formula $\mathsf{marked}(x)$ such that for every tree $t \in \mathsf{Tr}_{A\setminus\{\mathbb{O}_\ddagger,\mathbb{1}_\ddagger,\heartsuit\}}$ with precisely 2^{n+1} marked leaves we have: $\mathfrak{G}_0 \models \mathsf{marked}(t)$ if and only if the marked leaves of t are the leaves of some (unique) marked tile subtree of t.*

Proof. The formula $\mathsf{marked}(x)$ states that once we select any of the 2^{n+1} marked leaves, we can execute from the resulting tree some sequence in the language $h^{2^{n+1}-1}u^D\{\ell, r\}^{n+1}\Theta_0$. Formally, we define $\mathsf{marked}(x) = \forall y(m_\ddagger(x, y) \to \exists z : [h^{2^{n+1}-1}u^D\{\ell, r\}^{n+1}\Theta_0](y, z))$. Let us explain the intuition behind this. Assume that we select exactly one of the 2^{n+1} marked leaves of t, and let t' be the resulting tree. First, note that by executing the sequence $h^{2^{n+1}-1}$ from t', we replace each of the marked leaves of t' with the symbol $\heartsuit$, reaching a tree t''. Second, by executing u^D from t'' we reach (in case t contains a marked tile subtree) a tree t''', which has the form of a chain where the lowest leaf is labeled with $\mathbb{O}_\ddagger$ or $\mathbb{1}_\ddagger$ and all other leaves are labeled with $\heartsuit$. Next, we can "shrink" the chain t''' to the tree $\theta(X_\ddagger)$ by executing some sequence from $\{\ell, r\}^{n+1}$. To $\theta(X_\ddagger)$ we can finally apply the action $\theta \in \Theta_0 \subseteq \Sigma$. ◀ ◀

A *grid tree* is a tree t for which every leaf is inside a subtree of t that is a tile tree. The following lemma can be proven similarly as Lemma 10.

▶ **Lemma 11.** *One can compute in logspace a first-order formula $\mathsf{grid}(x)$ such that for all $t \in \mathsf{Tr}_A$ we have $\mathfrak{G}_0 \models \mathsf{grid}(t)$ if and only if t is a grid tree.*

A *marked grid tree* is a tree that can be obtained from a grid tree t by replacing exactly one tile subtree of t by some marked tile tree. A *selected grid tree* is a tree that can be obtained from a marked grid tree t by selecting *precisely one* marked leaf of t. For each $i \in [1, n + 1]$ we define the logspace computable FO-formula

$$\mathrm{bit}_i(x) = \exists y : [h^{2^{n+1}-1}u^D\{\ell, r\}^{i-1}r](x, y).$$

It is not hard to see that for every selected grid tree t with selected leaf λ we have that the i^{th} least significant bit of $\mathsf{lex}(\lambda)$ is 1 if and only if $\mathfrak{G}_0 \models \mathrm{bit}_i(t)$. Next, compute for each $\circ \in \{<, =\}$ in logspace a first-order formula $\varphi_\circ(x,y)$ such that for every two selected grid trees t_1 and t_2 with selected leaves λ_1 and λ_2 we have $\mathfrak{G}_0 \models \varphi_\circ(t_1,t_2)$ if and only if $\mathsf{lex}(\lambda_1) \circ \mathsf{lex}(\lambda_2)$. We define

$$\varphi_<(x,y) = \bigvee_{j\in[1,n+1]} \Big((\neg \mathrm{bit}_j(x) \wedge \mathrm{bit}_j(y)) \wedge \bigwedge_{1\le i<j} (\mathrm{bit}_i(x) \leftrightarrow \mathrm{bit}_i(y)) \Big)$$

and $\varphi_=(x,y) = \neg\varphi_<(x,y) \wedge \neg\varphi_<(y,x)$. Recall that the marked tile subtree of a marked grid tree t represents a θ-labeled grid element $(M,N) \in [0, 2^{2^n}-1]^2$ for some $\theta \in \Theta_0$. Let us define $M(t) = M$, $N(t) = N$, and $\Theta_0(t) = \theta$.

▶ **Lemma 12.** *One can compute in logspace first-order formulas $\varphi_\theta(x)$, $\varphi^i_M(x,x')$, $\varphi^i_N(x,x')$, where $\theta \in \Theta_0$ and $i \in \{0,1\}$ such that for all marked grid trees t and t' the following holds:*
(1) $\mathfrak{G}_0 \models \varphi_\theta(t)$ *if and only if* $\Theta_0(t) = \theta$ *and*
(2) $\mathfrak{G}_0 \models \varphi^i_Y(t,t')$ *(where $Y \in \{M,N\}$) if and only if $Y(t) + i = Y(t')$*

Proof. For point (1), let $\varphi_\theta(x) = \exists y : [m_\ddagger h^{2^{n+1}-1} u^D \{\ell, r\}^{n+1} \theta](x,y)$. For point (2), we only construct the formula $\varphi^1_M(x,x')$. For a selected grid tree z, the formula $l(z) = \exists u,v(h(z,u) \wedge \ell(u,v))$ expresses that the selected leaf is a left child. Then define

$$\varphi^1_M(x,y) = \exists x', y' (m_\ddagger(x,x') \wedge m_\ddagger(y,y') \wedge \varphi_=(x',y') \wedge \mathbb{O}_\ddagger(x',x') \wedge \not\Vdash_\ddagger(y',y') \wedge l(x') \wedge \psi_1 \wedge \psi_2).$$

Thus, we select a position $p \in [0, 2^n - 1]$ that is set to 0 (resp. 1) in the binary representation of $M(t)$ (resp. $M(t')$). The formula $\psi_1(x,y,x',y')$ is the conjunction

$$\forall z((m_\ddagger(x,z) \wedge \varphi_<(z,x') \wedge l(z)) \to \not\Vdash_\ddagger(z,z)) \quad \wedge \quad \forall z((m_\ddagger(y,z) \wedge \varphi_<(z,y') \wedge l(z)) \to \mathbb{O}_\ddagger(z,z)).$$

It expresses that each bit at some position that is smaller than p is set to 1 (resp. 0) in $M(t)$ (resp. $M(t')$). Finally, the formula $\psi_2(x,y,x',y')$ is

$$\forall u,v \ ((m_\ddagger(x,u) \wedge m_\ddagger(y,v) \wedge \varphi_=(u,v) \wedge \varphi_<(x',u) \wedge l(u)) \to (\not\Vdash_\ddagger(u,u) \leftrightarrow \not\Vdash_\ddagger(v,v))).$$

It expresses that the binary representations of $M(t)$ and $M(t')$ agree on each position $> p$. ◀ ◀

Recall that $w = \theta_0 \cdots \theta_{n-1}$. We define the formula $\mathsf{sol}(x)$ as the conjunction of $\mathsf{grid}(x)$ and the following formulas, where $\mathsf{mark}(z_1,z_2)$ abbreviates $m_\ddagger^{2^{n+1}}(z_1,z_2) \wedge \mathsf{marked}(z_2)$:

- Grid element $(j,0)$ is labeled by θ_j for all $j \in [0, n-1]$:

$$\exists y_0,\ldots,y_{n-1}\Big(\bigwedge_{j\in[0,n-1]} (\mathsf{mark}(x,y_j) \wedge \varphi_{\theta_j}(y_j)) \ \wedge \ \forall z(m_\ddagger(y_0,z) \to \mathbb{O}_\ddagger(z,z)) \wedge \bigwedge_{j\in[1,n-1]} (\varphi^1_M(y_{j-1},y_j) \wedge \varphi^0_N(y_{j-1},y_j))\Big)$$

- If we mark a tile subtree of x that corresponds to the grid element (M,N) and $M < 2^{2^n}-1$, a tile subtree of x that corresponds to $(M+1,N)$ satisfies the horizontal matching relation:

$$\forall y((\mathsf{mark}(x,y) \wedge \exists z(m_\ddagger(y,z) \wedge \mathbb{O}_\ddagger(z,z) \wedge l(z))) \to \exists y'(\mathsf{mark}(x,y') \wedge \varphi^1_M(y,y') \wedge \varphi^0_N(y,y') \wedge \bigvee_{(\theta,\theta')\in\mathbb{H}_0} (\varphi_\theta(y) \wedge \varphi_{\theta'}(y'))))$$

- Analogously, we express that the vertical matching relation is respected and for each grid element there is at most one tile type.

It follows by construction that $\mathsf{Sol}_{2^{2^n}}(S_0, w) \neq \emptyset$ if and only if $\mathfrak{G}_0 \models \exists x : \mathsf{sol}(x)$. Thus, the first-order theory of $\mathfrak{G}_0$ is indeed 2NEXP-hard under logspace reductions.

We conclude with a non-elementary lower bound for ground tree rewrite graphs with an additional unary predicate. For a GTRS $\mathcal{R} = (A, \Sigma, R)$ and a set of trees $L \subseteq \mathsf{Tr}_A$, we denote with $(\mathfrak{G}(\mathcal{R}), L)$ the structure that results from the labelled graph $\mathfrak{G}(\mathcal{R})$ by adding the set L as an additional unary predicate. Note that if L is a regular set of trees, then $(\mathfrak{G}(\mathcal{R}), L)$ is a tree-automatic structure, and hence has a decidable first-order theory. On the other hand, a reduction from satisfiability for first-order logic over binary words (see [11]) shows:

▶ **Theorem 13.** *There exists a fixed GTRS $\mathcal{R}_1 = (A, \Sigma, R)$ and a fixed regular tree language $L \subseteq \mathsf{Tr}_A$ such that the first-order theory of $(\mathfrak{G}(\mathcal{R}_1), L)$ is non-elementary.*

References

1 L. Berman. The complexity of logical theories. *Theoret. Comput. Sci.*, 11:71–77, 1980.
2 E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem.* Springer, 2001.
3 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. CONCUR'97*, LNCS 1243, pages 135–150. Springer, 1997.
4 W. S. Brainerd. Tree generating regular systems. *Inform. and Control*, 14(2):217–231, 1969.
5 A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
6 K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Logic*, 48(1):1–79, 1990.
7 J.-L. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata: Classification and connection with rewrite systems. *Theor. Comput. Sci.*, 127(1):69–98, 1994.
8 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. LICS'90*, pages 242–248. IEEE Computer Society, 1990.
9 J. Ferrante and C. Rackoff. *The Computational Complexity of Logical Theories.* Number 718 in Lecture Notes in Mathematics. Springer, 1979.
10 S. Göller and A. W. Lin. The complexity of verifying ground tree rewrite systems. In *Proc. LICS 2011*, pages 279-288. IEEE Computer Society, 2011.
11 S. Göller and M. Lohrey. The first-order theory of ground tree rewrite graphs. arXiv.org, 2011. `http://arxiv.org/abs/1107.0919`.
12 D. Kuske and M. Lohrey. Automatic structures of bounded degree revisited. In *Proc. CSL 2009*, LNCS 5771, pages 364–378. Springer, 2009.
13 C. Löding. *Infinite Graphs Generated by Tree Rewriting.* PhD thesis, RWTH Aachen, 2003.
14 D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
15 L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, Department of Electrical Engineering, MIT, 1974.
16 S. Tison. Fair termination is decidable for ground systems. In *Proc. RTA'89*, LNCS 355, pages 462–476. Springer, 1989.
17 A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches.* PhD thesis, LFCS, School of Informatics, University of Edinburgh, 2010.
18 H. Vogel. Turing machines with linear alternation, theories of bounded concatenation and the decision problem of first-order theories,. *Theoret. Comput. Sci.*, 23:333–337, 1983.

19 I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proc. FSTTCS 2000*, LNCS 1974, pages 127–138. Springer, 2000.

20 I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

Layer Systems for Proving Confluence*

Bertram Felgenhauer, Harald Zankl, and Aart Middeldorp

Institute of Computer Science, University of Innsbruck, Austria

Abstract

We introduce layer systems for proving generalizations of the modularity of confluence for first-order rewrite systems. Layer systems specify how terms can be divided into layers. We establish structural conditions on those systems that imply confluence. Our abstract framework covers known results like many-sorted persistence, layer-preservation and currying. We present a counterexample to an extension of the former to order-sorted rewriting and derive new sufficient conditions for the extension to hold.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Term rewriting, Confluence, Modularity, Persistence

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.288

1 Introduction

In this paper we revisit the celebrated modularity result of confluence, due to Toyama [13]. It states that the union of two confluent rewrite systems is confluent, provided the participating rewrite systems do not share function symbols. This result has been reproved several times, using category theory [10], ordered completion [6], and decreasing diagrams [14]. In practice, modularity is of limited use. More useful techniques, in the sense that rewrite systems can be decomposed into smaller systems that share function symbols and rules, are based on type introduction [2], layer-preservation [11], and commutativity [12].

Type introduction [16] restricts the set of terms that have to be considered to the well-typed terms according to any many-sorted type discipline which is compatible with the rewrite system under consideration. A property of rewrite systems for which type introduction is correct is called persistent and Aoto and Toyama [2] showed that confluence is persistent. In [1] they extended the latter result by considering an order-sorted type discipline. However, we show that the conditions imposed in [1] are not sufficient for confluence.

The proofs in [11] and [1, 2] are adaptations of the proof of Toyama's modularity result by Klop *et al.* [9]. A more complicated proof using concepts from [9] has been given by Kahrs, who showed in [7] that confluence is preserved under currying [8]. In this paper we introduce *layer systems* as a common framework to capture the results of [2, 7, 11, 13] and to identify appropriate conditions to restore the persistence of confluence for order-sorted rewriting [1]. Layer systems identify the parts that are available when decomposing terms. The key proof idea remains the same. We treat each such layer independently from the others where possible, and deal with interactions between layers separately. The main advantage of and motivation for our proof is that the result becomes reusable; instead of checking every detail of a complex proof, we have to check a couple of comparatively simple, structural conditions on layer systems instead.

The remainder of this paper is organized as follows. In the next section we recall preliminaries. Section 3 presents a counterexample to [1, Theorem 4.12]. Layer systems are

* This research was supported by the Austrian Science Fund (FWF) P22467-N23.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 288–299

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

introduced in Section 4 and in Section 5 we develop conditions that guarantee the confluence of rewrite systems if a compatible layer system exists. In Section 6 we instantiate the abstract setting to cover the applications mentioned earlier before concluding in Section 7. Proofs can be found in an accompanying technical report [5] but key lemmata are presented to indicate the overall proof idea.

2 Preliminaries

We assume familiarity with rewriting [3]. A signature consists of function symbols $\mathcal{F}$ and variables $\mathcal{V}$. Each $f \in \mathcal{F}$ has a fixed arity $\mathrm{arity}(f)$. We assume that $\mathcal{V}$ is infinite. The set of terms over the signature $\langle \mathcal{F}, \mathcal{V} \rangle$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. The sets of variables and function symbols occurring in a term t are denoted by $\mathcal{V}\mathsf{ar}(t)$ and $\mathcal{F}\mathsf{un}(t)$, respectively. The positions of a term t are strings of natural numbers, ϵ for the root, and ip if $t = f(t_1, \ldots, t_i, \ldots, t_n)$ and p is a position of t_i. $\mathcal{P}\mathsf{os}(t)$ is the set of all positions of t. For positions p, q, r we write $p < q$ if p is a proper prefix of q, $p \leqslant q$ if $p < q$ or $p = q$, $p \parallel q$ if neither $p < q$ nor $q \leqslant p$. If $p \leqslant q$ then $r = q \setminus p$ denotes the unique position such that $pr = q$. Given terms t and s, $t|_p$ is the subterm at position p of t, $t(p)$ is the root symbol of $t|_p$ and $t[s]_p$ denotes the result of replacing $t|_p$ by s in t. For $X \subseteq \mathcal{F} \cup \mathcal{V}$, we let $\mathcal{P}\mathsf{os}_X(t) = \{p \in \mathcal{P}\mathsf{os}(t) \mid t(p) \in X\}$. A substitution is a map $\sigma \colon \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ which extends homomorphically to terms.

A rewrite rule is a pair of terms $(\ell, r) \in \mathcal{T}(\mathcal{F}, \mathcal{V})^2$, written $\ell \to r$ such that $\ell \notin \mathcal{V}$ and $\mathcal{V}\mathsf{ar}(\ell) \supseteq \mathcal{V}\mathsf{ar}(r)$. A term rewrite system (TRS) is a set of rewrite rules. The rewrite relation induced by a TRS $\mathcal{R}$ is denoted $\to_{\mathcal{R}}$. We write $\leftarrow$, $\to^=$, $\to^+$ and $\to^*$ to denote the inverse, the reflexive closure, the transitive closure and the reflexive and transitive closure of a relation $\to$, respectively. A relation $\to$ is confluent if ${}^*\!\leftarrow \cdot \to^* \subseteq \to^* \cdot {}^*\!\leftarrow$ and terminating if $\to^+$ is well-founded. A TRS $\mathcal{R}$ inherits these properties from $\to_{\mathcal{R}}$.

Next we recall *many-sorted* terms. Let $\mathcal{S}$ be a set of sorts. A signature $\langle \mathcal{F}, \mathcal{V} \rangle$ is S-sorted, if every n-ary $f \in \mathcal{F}$ is equipped with a sort declaration $\alpha_1 \times \cdots \times \alpha_n \to \alpha$ where $\alpha_1, \ldots, \alpha_n, \alpha \in \mathcal{S}$ and every $x \in \mathcal{V}$ has exactly one sort $\alpha \in S$. We let $\mathcal{V}_\alpha = \{x \in \mathcal{V} \mid x \text{ has sort } \alpha\}$, and require that $\mathcal{V}_\alpha$ is infinite for all $\alpha \in S$. The set of many-sorted terms, $\mathcal{T}_{\mathcal{S}}(\mathcal{F}, \mathcal{V})$, is the union of the sets $\mathcal{T}_\alpha(\mathcal{F}, \mathcal{V})$ for $\alpha \in \mathcal{S}$ that are inductively defined as follows: $\mathcal{V}_\alpha \subseteq \mathcal{T}_\alpha(\mathcal{F}, \mathcal{V})$ and $f(t_1, \ldots, t_n) \in \mathcal{T}_\alpha(\mathcal{F}, \mathcal{V})$ whenever $f \in \mathcal{F}$ has sort declaration $\alpha_1 \times \cdots \times \alpha_n \to \alpha$ and $t_i \in \mathcal{T}_{\alpha_i}(\mathcal{F}, \mathcal{V})$ for all $1 \leqslant i \leqslant n$. If $t \in \mathcal{T}_\alpha(\mathcal{F}, \mathcal{V})$ for some $\alpha \in \mathcal{S}$ then we say that t has sort α and write $\mathrm{sort}(t) = \alpha$. A many-sorted (or $\mathcal{S}$-sorted) TRS consists of an $\mathcal{S}$-sorted signature and a set $\mathcal{R}$ of rewrite rules between many-sorted terms of the same sort. A property P of TRSs is called *persistent* if a many-sorted TRS has the property P if and only if its underlying (unsorted) TRS has the property P. To obtain $(\mathcal{S}, \succeq)$-*order-sorted* terms, we equip the sorts of an $\mathcal{S}$-sorted signature with a partial order $\succeq$. Each set $\mathcal{T}_\alpha(\mathcal{F}, \mathcal{V})$ of $\mathcal{S}$-sorted terms of sort α is extended with $f(t_1, \ldots, t_n)$ whenever $f \in \mathcal{F}$ has sort declaration $\alpha_1 \times \cdots \times \alpha_n \to \alpha$ and, for all $1 \leqslant i \leqslant n$, $t_i \in \mathcal{T}_{\beta_i}(\mathcal{F}, \mathcal{V})$ for some sort β_i with $\alpha_i \succeq \beta_i$. A variable occurrence $x = t|_p \in \mathcal{V}_\alpha$ is called *strictly bound* if $p = \epsilon$ or its sort exactly matches that of its context, i.e., $\alpha_i = \beta_i$ above.

3 A Counterexample

The result claimed in [1] states that the underlying unsorted TRS $\mathcal{R}$ of a confluent $(\mathcal{S}, \succeq)$-order-sorted TRS $\mathcal{R}$ is confluent (on arbitrary terms) provided the strict part of $\succeq$ is well-founded, $\mathrm{sort}(\ell) \succeq \mathrm{sort}(r)$ and variable occurrences are strictly bound in ℓ and r for every rewrite rule $\ell \to r \in \mathcal{R}$. Below we give a counterexample to this claim (we derive a corrected criterion in Section 6).

▶ **Example 3.1.** Consider the TRS $\mathcal{R}$ consisting of $\mathsf{c}(x) \to x$ and the following rewrite rules:

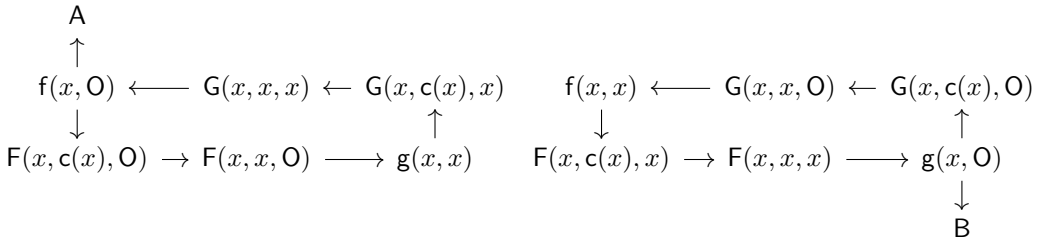
$$\mathsf{f}(x,y) \to \mathsf{F}(x,\mathsf{c}(x),y) \quad \mathsf{f}(x,\mathsf{O}) \to \mathsf{A} \quad \mathsf{F}(x,y,\mathsf{O}) \to \mathsf{g}(x,y) \quad \mathsf{F}(x,y,y) \to \mathsf{g}(x,\mathsf{O})$$
$$\mathsf{g}(x,y) \to \mathsf{G}(x,\mathsf{c}(x),y) \quad \mathsf{g}(x,\mathsf{O}) \to \mathsf{B} \quad \mathsf{G}(x,y,\mathsf{O}) \to \mathsf{f}(x,y) \quad \mathsf{G}(x,y,y) \to \mathsf{f}(x,\mathsf{O})$$

We take $\mathcal{S} = \{0,1,2\}$ as sorts with $1 \succeq 0$ and the signature given by

$$\mathsf{f},\mathsf{g} : 0 \times 1 \to 2 \quad \mathsf{A},\mathsf{B} : 2 \quad \mathsf{c} : 0 \to 1 \quad \mathsf{F},\mathsf{G} : 0 \times 1 \times 1 \to 2 \quad \mathsf{O}, y : 1 \quad x : 0$$

All rules except for $\mathsf{c}(x) \to x$ are many-sorted and sort-preserving, while $\mathrm{sort}(\mathsf{c}(x)) = 1 \succeq 0 = \mathrm{sort}(x)$ for $\mathsf{c}(x) \to x$. Hence $\mathcal{R}$ satisfies the aforementioned constraints from [1].

The rewrite relation defined by $\mathcal{R}$ is essentially finite: since c does not occur in any left-hand side of any rewrite rule but $\mathsf{c}(x) \to x$, we can apply this rule to any term and remove all occurrences of c, without interfering with any other future rewrite steps. The remaining order-sorted terms are flat terms (i.e., of depth 1 or 0), and there are only finitely many of those up to variable renaming. The interesting part of the rewrite relation looks as follows:



Note that x cannot be replaced by O anywhere, since this would make the terms ill-sorted. Hence $\mathcal{R}$ is confluent on $\mathcal{T}_{\mathcal{S}}(\mathcal{F},\mathcal{V})$. However, the corresponding unsorted TRS $\mathcal{R}$ is not confluent, since $\mathsf{A} \leftarrow \mathsf{f}(\mathsf{O},\mathsf{O}) \to^* \mathsf{g}(\mathsf{O},\mathsf{O}) \to \mathsf{B}$ for normal forms A and B.

4 Layer Systems

The existing proofs of modularity and its generalizations work by dividing terms into a top context (or native part) and principal subterms (aliens). This process can be characterized by the set of terms that are allowed as top contexts. We call this set a *layer system*. Layers are contexts, which can be represented by terms with holes. We use a fresh constant $\Box$ to denote such holes. Terms with holes can be merged.

▶ **Definition 4.1.** The partial function merge $\sqcup\colon \mathcal{T}(\mathcal{F} \cup \{\Box\},\mathcal{V})^2 \to \mathcal{T}(\mathcal{F} \cup \{\Box\},\mathcal{V})$ satisfies:

- $\Box \sqcup t = t \sqcup \Box = t$ for $t \in \mathcal{T}(\mathcal{F} \cup \{\Box\},\mathcal{V})$,
- $x \sqcup x = x$ for $x \in \mathcal{V}$,
- $f(s_1,\dots,s_n) \sqcup f(t_1,\dots,t_n) = f(u_1,\dots,u_n)$ if $f \in \mathcal{F}$ and $s_i \sqcup t_i = u_i$ for $1 \leqslant i \leqslant n$.

▶ **Example 4.2.** We have $\mathsf{f}(\mathsf{h}(\Box),\Box) \sqcup \mathsf{f}(\Box,\mathsf{g}(y)) = \mathsf{f}(\mathsf{h}(\Box),\mathsf{g}(y))$ but $\mathsf{f}(\mathsf{h}(\Box),\mathsf{g}(x)) \sqcup \mathsf{f}(\Box,\mathsf{g}(y))$ and $\mathsf{f}(\mathsf{h}(\Box),\Box) \sqcup \mathsf{f}(\mathsf{g}(\Box),\Box)$ are undefined (different non-hole symbols cannot be merged).

▶ **Definition 4.3.** A *layer system* is a set of terms $\mathbb{L} \subseteq \mathcal{T}(\mathcal{F} \cup \{\Box\},\mathcal{V})$ satisfying:

1. $\Box \in \mathbb{L}$, $\mathcal{V} \subseteq \mathbb{L}$, and $f(\Box,\dots,\Box) \in \mathbb{L}$ for $f \in \mathcal{F}$.
2. $\mathbb{L}$ allows *replacing holes by variables and vice versa*: If $t[\Box]_p \in \mathbb{L}$ then $\{x \in \mathcal{V} \mid t[x]_p \in \mathbb{L}\}$ is an infinite set. Furthermore, if $t[x]_p \in \mathbb{L}$ and $x \in \mathcal{V}$ then $t[\Box]_p \in \mathbb{L}$.
3. $\mathbb{L}$ is closed under *merging at function positions in* $\mathcal{F}$: If $s,t \in \mathbb{L}$, $p \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(t)$, and $u := t|_p \sqcup s$ is defined then $t[u]_p \in \mathbb{L}$.
4. $\mathbb{L}_{\mathcal{T}} := \mathbb{L} \cap \mathcal{T}(\mathcal{F},\mathcal{V})$ is closed under $\to_{\mathcal{R}}$: If $s \in \mathbb{L}_{\mathcal{T}}$ and $s \to_{\mathcal{R}} t$ then $t \in \mathbb{L}_{\mathcal{T}}$.

Requiring that $\mathbb{L}$ is closed under $\to_{\mathcal{R}}$ would imply Definition 4.3(4), but the weaker condition helps applications. For example, in the case of order-sorted persistence we do not have to worry about holes at positions with incompatible types, simplifying the proof that order-sorted terms form a layer system under certain conditions on $\mathcal{R}$ (cf. Theorem 6.2).

For pretty much the same reason, we do not require that holes can be replaced by any variable in Definition 4.3(2)—for example, in the order-sorted case, we would otherwise have to treat variables as essentially untyped, complicating that application.

Definition 4.3(3) allows one to merge two layers into a larger one. This is essential to obtain a *least layered representation* (cf. Theorem 5.2(4)) while Definition 4.3(1) ensures that any term can be layered (cf. Lemma 4.10(1)), by starting a new layer at every symbol.

We fix some layer system $\mathbb{L}$ that we want to use for layering terms. A key element of our proof is that this layering is made explicit using the notion of split terms.

▶ **Definition 4.4.** A *split term* $\hat{t}$ is a pair $\hat{t} = \langle t, P\rangle$ made of a term $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ and a set of so-called *special* positions $P \subseteq \mathcal{P}\mathsf{os}(t)$ of $\hat{t}$. We write $P_{\hat{t}}$ for the special positions of $\hat{t}$. We call t the *underlying term* of $\hat{t}$ and often denote it by t. A special position is called *principal* if it is a minimal special position with respect to $<$. A split term $\langle t, P\rangle$ is *simple* if $P = \varnothing$, *special* if $\epsilon \in P$ and *normal* if $\epsilon \notin P$. The set of split terms is denoted by $\widehat{\mathcal{T}}(\mathcal{F},\mathcal{V})$.

We identify a term t with the unique corresponding simple split term $\langle t, \varnothing\rangle$.

▶ **Definition 4.5.** We define split counterparts to standard operations on terms.

subterm $\hat{t}|_p := \langle t|_p, \{q \setminus p \mid q \in P_{\hat{t}} \text{ with } q \geqslant p\}\rangle$. Note that if p is special then $\hat{t}|_p$ is special.

modifying special positions at the root $\hat{t}|_- := \langle t, P_{\hat{t}} \setminus \{\epsilon\}\rangle$ and $\epsilon(\hat{t}) := \langle t, P_{\hat{t}} \cup \{\epsilon\}\rangle$. We let $\hat{t}|_{p-} := (\hat{t}|_p)|_-$ and we will also use the notation $\hat{t}|_{p*}$ to denote one of $\hat{t}|_p$ or $\hat{t}|_{p-}$. If p is a principal position of $\hat{t}$, then $\hat{t}|_{p-}$ is a *principal subterm* of $\hat{t}$.

replacing subterms $\hat{t}[\hat{s}]_p := \langle t[s]_p, \{q \mid q \in P_{\hat{t}} \text{ with } q \not> p\} \cup \{pq \mid q \in P_{\hat{s}}\}\rangle$. For a set of pairwise parallel positions $Q \subseteq \mathcal{P}\mathsf{os}(t)$ we also write $\hat{t}[\hat{s}_q]_{q\in Q}$ for the split term obtained from $\hat{t}$ by replacing $t|_q$ by $\hat{s}_q$ for each $q \in Q$.

substitution If $\sigma\colon \mathcal{V} \to \widehat{\mathcal{T}}(\mathcal{F},\mathcal{V})$ then $\sigma(\hat{t}) = \hat{t}[\sigma(t|_p)]_{p\in\mathcal{P}\mathsf{os}_{\mathcal{V}}(t)}$.

Next we introduce two natural ways to represent split terms.

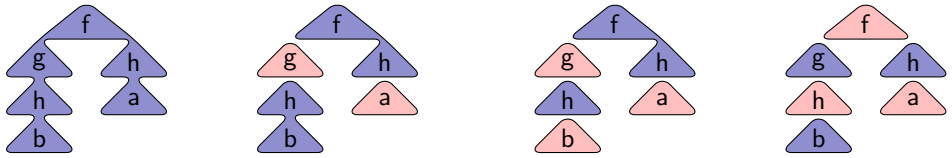
▶ **Definition 4.6.** A *split context* is a split term $\hat{C} = \langle C, P\rangle \in \widehat{\mathcal{T}}(\mathcal{F} \cup \{\Box\},\mathcal{V})$ such that $P = \{p \in \mathcal{P}\mathsf{os}(C) \mid C|_p = \Box\}$. We define $\hat{C}$ as the unique split context with underlying term C. For a split context $\hat{C}$ with n holes and normal terms $\hat{t}_i$ we denote by $\hat{C}[\![\hat{t}_1,\dots,\hat{t}_n]\!]$ the split term obtained by replacing the holes in $\hat{C}$ by $\hat{t}_1$ to $\hat{t}_n$ from left to right.

It is easy to see that any split term $\hat{t}$ can be uniquely represented as $\hat{t} = \hat{C}[\![\hat{t}_1,\dots,\hat{t}_n]\!]$. The corresponding split context is the *top layer* of $\hat{t}$.

▶ **Definition 4.7.** Let $\hat{t}$ be a split term. The *top layer* $\hat{L}(\hat{t})$ of $\hat{t}$ is obtained by replacing all its principal subterms by $\Box$. It is a split context as all remaining special positions are holes. We write $L(\hat{t})$ for the corresponding unsplit context. The *rank* of a split term $\hat{t} = \hat{C}[\![\hat{t}_1,\dots,\hat{t}_n]\!]$ is defined recursively by $\operatorname{rank}(\hat{t}) = 1 + \max\{0, \operatorname{rank}(\hat{t}_i) \mid 1 \leqslant i \leqslant n\}$.

The rank is useful for inductive proofs since principal subterms have smaller rank than the term itself. Next we use layer systems to restrict how terms can be split.

▶ **Definition 4.8.** A split term $\hat{t}$ is *layered according to* $\mathbb{L}$ or just *layered* if $L(\hat{t}) \in \mathbb{L}$ and $L(\hat{t}|_{p-}) \in \mathbb{L}$ for all $p \in P_{\hat{t}}$. The *layers* of $\hat{t}$ are the unsplit contexts $L(\hat{t})$ and $L(\hat{t}|_{p-})$ for $p \in P_{\hat{t}}$. For a layered term $\hat{s}$, a layer $s' \in \mathbb{L}_{\mathcal{T}}$ and a substitution σ we say that $\sigma(s')$ *designates* $\hat{s}$ if $\sigma(s') = \hat{s}$, $\sigma(x) \neq x$ implies $\sigma(x)$ is special, and $\sigma(x) = x$ for $x \in \mathcal{V}\mathsf{ar}(s)$.



(a) $P_{\hat{t}} = \varnothing$ **(b)** $P_{\hat{t}} = \{1, 11, 21\}$ **(c)** $P_{\hat{t}} = \{1, 11, 111, 21\}$ **(d)** $P_{\hat{t}} = \mathcal{P}\mathsf{os}(t)$

Figure 1 Some split terms for $t = \mathsf{f}(\mathsf{g}(\mathsf{h}(\mathsf{b})), \mathsf{h}(\mathsf{a}))$.

Designation is similar to the division of terms into a cap and an alien substitution in [6].

▶ **Example 4.9.** Figure 1 shows some split terms. Layers are colored alternately between (dark) blue and (light) red with the top layer (if non-empty) marked blue. Figure 1(a) depicts a simple term. The term in Figure 1(d) is special, the others are normal. In Figure 1(b) the principal positions are 1 and 21 with $\langle \mathsf{g}(\mathsf{h}(\mathsf{b})), \{1\}\rangle$ and $\langle \mathsf{a}, \varnothing\rangle$ as the corresponding principal subterms. The ranks of the terms in Figures 1(c) and 1(d) are 4 and 5 and the top layers are $\langle \mathsf{f}(\Box, \mathsf{h}(\Box)), \{1, 21\}\rangle$ and $\langle \Box, \{\epsilon\}\rangle$, respectively. For suitable $\mathbb{L}$, the term in Figure 1(b) is designated by $\sigma(\mathsf{f}(x, \mathsf{h}(y)))$ with $\sigma(x) = \langle \mathsf{g}(\mathsf{h}(\mathsf{b})), \{\epsilon, 1\}\rangle$ and $\sigma(y) = \langle \mathsf{a}, \{\epsilon\}\rangle$.

▶ **Lemma 4.10.** *We state some fundamental properties of layered terms.*

1. *For any term s the split term $\langle s, \mathcal{P}\mathsf{os}(s)\rangle$ is layered.*
2. *If $\hat{t}$ is layered and $p \in P_{\hat{t}}$ then $\hat{t}|_p$ and $\hat{t}|_{p-}$ are layered.*
3. *If $\hat{t}_1 = \langle t, P_1\rangle$ and $\hat{t}_2 = \langle t, P_2\rangle$ are layered then $\langle t, P_1 \cap P_2\rangle$ is layered.*

As a consequence of Lemma 4.10, every term t has a *least layered representation* $\hat{t}_{\mathbb{L}}$, which can be defined as $\hat{t}_{\mathbb{L}} := \langle t, \bigcap\{P \mid \langle t, P\rangle \text{ is layered according to } \mathbb{L}\}\rangle$.

▶ **Example 4.11.** Consider the $\{0, 1\}$-sorted signature with $\mathsf{f} : 0 \times 1 \to 0$, $\mathsf{g} : 0 \to 1$, $\mathsf{h} : 1 \to 1$, $\mathsf{a} : 0$, $\mathsf{b} : 1$, and let $\mathbb{L} := \mathcal{T}_{\mathcal{S}}(\mathcal{F} \cup \{\Box\}, \mathcal{V})$ where holes can appear anywhere in terms, both with sort 0 and 1. The term in Figure 1(a) is not layered according to $\mathbb{L}$, the others are. Figure 1(b) depicts $\hat{t}_{\mathbb{L}}$, i.e., a new layer starts if and only if a subterm does not match the sort of the context.

▶ **Lemma 4.12.** *Any layered term $\hat{s}$ is designated by some $\sigma(s')$ with $s' = L(\hat{s})[\vec{x}] \in \mathbb{L}$, $x_i \in \mathcal{V}$.*

Designations will be used to simulate so-called *outer* rewrite steps (cf. Definition 5.1) by rewrite steps inside $\mathbb{L}_{\mathcal{T}}$. Lemma 4.12 is a first step towards this goal.

5 Layer Systems for Confluence

In this section we first turn to rewriting split terms and then impose conditions on layer systems such that a TRS $\mathcal{R}$ is confluent if it is confluent on $\mathbb{L}_{\mathcal{T}}$. The overall idea is to start with rewrite sequences on terms, turn them to rewrite sequences on layered terms, which enjoy confluence, and map the resulting joining sequences back to (unsplit) terms.

▶ **Definition 5.1.** Let $\hat{s}$ and $\hat{t}$ be split terms. A *rule step* $\hat{s} \to_{p,\ell\to r} \hat{t}$ satisfies $\ell \to r \in \mathcal{R}$, $\hat{s}|_{p-} = \sigma(\ell)$ and $\hat{t} = \hat{s}[\sigma(r)]_p$ for some substitution σ. We also write $\hat{s} \to_{\mathcal{R}} \hat{t}$. A rule step $\hat{s} \to_{p,\ell\to r} \hat{t}$ is *inner*, $\hat{s} \to_i \hat{t}$, if $p \geqslant q$ for some $q \in P_{\hat{s}}$, and *outer*, $\hat{s} \to_o \hat{t}$, otherwise. A *fusion step* $\hat{s} \rightsquigarrow \hat{t}$ satisfies $s = t$ and $P_{\hat{s}} \supseteq P_{\hat{t}}$. It is *proper*, $\hat{s} \overset{\neq}{\rightsquigarrow} \hat{t}$, if $P_{\hat{s}} \supsetneq P_{\hat{t}}$. The union of $\to_{\mathcal{R}}$ and $\rightsquigarrow$ is denoted by $\rightsquigarrow_{\mathcal{R}}$.

Rule steps correspond to normal rewrite steps. Since ℓ and r are (simple) terms, matching is constrained to a single layer of a term, and the result of the rewrite step will only modify that layer and possibly permute, erase or duplicate special subterms below the layer.

Fusion steps, on the other hand, allow adjacent layers to be merged into a single one. Note that even when starting from a least layered representation, $\hat{s}_{\mathbb{L}} \to_{\mathcal{R}} \hat{t}$ may result in a term $\hat{t}$ that allows a proper fusion step. In the classical modularity setting this may happen after applying a collapsing rule. In many previous proofs from the literature, fusion was implicit. By making fusion explicit, we can capture the phenomena that destroy modularity more precisely, and, perhaps more importantly, we can delay fusion in joining rewrite sequences until we can comfortably deal with it.

▶ **Lemma 5.2.** *Rewriting on split terms has the following properties.*

1. *If $\hat{s} \rightsquigarrow_{\mathcal{R}} \hat{t}$ then $\sigma(\hat{s}) \rightsquigarrow_{\mathcal{R}} \sigma(\hat{t})$ for any (split) substitution σ.*
2. *If $\hat{s} \to_o \hat{t}$ then every principal subterm of $\hat{t}$ is a principal subterm of $\hat{s}$.*
3. *If $\hat{s} \rightsquigarrow_{\mathcal{R}} \hat{t}$ then $\mathrm{rank}(\hat{s}) \geqslant \mathrm{rank}(\hat{t})$.*
4. *If $\hat{t}$ is layered then $\hat{t} \rightsquigarrow \hat{t}_{\mathbb{L}}$.*

We extend fusion steps to substitutions: $\sigma \rightsquigarrow \tau$ if $\sigma(x) \rightsquigarrow \tau(x)$ for all $x \in \mathcal{V}$.

▶ **Lemma 5.3.** *Let $\hat{s}$ and $\hat{t}$ be split terms. If $\hat{s} \rightsquigarrow \hat{s}'$ then $\hat{t}[\hat{s}]_p \rightsquigarrow \hat{t}[\hat{s}']_p$ for any $p \in \mathcal{P}\mathsf{os}(t)$. For split substitutions σ and σ', if $\sigma \rightsquigarrow \sigma'$ then $\sigma(\hat{t}) \rightsquigarrow \sigma'(\hat{t})$.*

We impose suitable additional constraints on the layer system $\mathbb{L}$ to prove that if $\to_{\mathcal{R}}$ is confluent on $\mathbb{L}_{\mathcal{T}}$ then $\to_{\mathcal{R}}$ is confluent. The converse is trivial, cf. Definition 4.3(4).

▶ **Definition 5.4.** A layer system $\mathbb{L}$ is *weakly consistent* with a TRS $\mathcal{R}$ if for all $\ell \to r \in \mathcal{R}$:

1. $\ell \in \mathbb{L}$,
2. $t[t|_{pq'}]_{pq} \in \mathbb{L}$ whenever $t \in \mathbb{L}$, $\ell|_q = \ell|_{q'} \in \mathcal{V}$ and $t|_p$ can be obtained from ℓ by replacing variables by terms—where different instances of the same variable may be replaced by different terms. (For left-linear TRSs this means that ℓ matches $t|_p$.)

Definition 5.4(2) always holds for left-linear systems, but both conditions are essential for general systems. The next example shows how Definition 5.4(1,2) may fail for non-confluent systems.

▶ **Example 5.5.** Consider the non-confluent $\mathcal{R} = \{\mathsf{f}(\mathsf{f}(x)) \to \mathsf{a}, \mathsf{f}(\mathsf{f}(x)) \to \mathsf{b}\}$. Then $\mathbb{L} = \mathcal{V} \cup \{\Box, \mathsf{a}, \mathsf{b}, \mathsf{f}(\Box)\} \cup \{\mathsf{f}(x) \mid x \in \mathcal{V}\}$ is a layer system such that $\to_{\mathcal{R}}$ is confluent on $\mathbb{L}_{\mathcal{T}}$. We have $\mathsf{f}(\mathsf{f}(x)) \notin \mathbb{L}$, so it violates Definition 5.4(1).

Next consider $\mathcal{R} = \{\mathsf{f}(x, x) \to \mathsf{a}, \mathsf{f}(\mathsf{a}, x) \to \mathsf{b}\}$ with $\mathbb{L} = \{\mathsf{f}(x, y), \mathsf{f}(\mathsf{a}, x), x, \mathsf{a}, \mathsf{b} \mid x, y \in \mathcal{V} \cup \{\Box\}\}$. This is a layer system satisfying Definition 5.4(1) and $\to_{\mathcal{R}}$ is confluent on $\mathbb{L}_{\mathcal{T}}$. However, since $t = \ell' = \mathsf{f}(\mathsf{a}, x)$ can be obtained from $\ell = \mathsf{f}(x, x)$ by replacing the first x by a, by Definition 5.4(2), we should have $t[\ell'|_1]_2 = \mathsf{f}(\mathsf{a}, \mathsf{a}) \in \mathbb{L}$.

As a final example, consider $\mathcal{R} = \{\mathsf{f}(x, x, y) \to \mathsf{g}(x, y), \mathsf{f}(x, y, z) \to \mathsf{a}\}$, and $\mathbb{L} = \{\mathsf{f}(x, y, z) \mid x \in \mathcal{V}_1 \cup \{\Box\}, y \in \mathcal{V}_2 \cup \{\Box\}, z \in \mathcal{V} \cup \{\mathsf{a}, \Box\}\} \cup \{\mathsf{a}, x, \mathsf{g}(x, y) \mid x, y \in \mathcal{V} \cup \{\Box\}\}$, where $\mathcal{V}_1, \mathcal{V}_2 \subseteq \mathcal{V}$ are disjoint and infinite. Again, $\to_{\mathcal{R}}$ is confluent on $\mathbb{L}_{\mathcal{T}}$ but not on $\mathcal{T}(\mathcal{F}, \mathcal{V})$. In this case, $s = \mathsf{f}(\mathsf{a}, \mathsf{a}, \mathsf{a})$ has $\hat{s}_{\mathbb{L}} = \langle s, \{1, 2\}\rangle$ and $\hat{s}_{\mathbb{L}} \to_{\mathcal{R}} \hat{t} = \langle \mathsf{g}(\mathsf{a}, \mathsf{a}), \{1\}\rangle$, but $\hat{t}$ is not layered. In this case, $\mathbb{L}$ violates Definition 5.4(2); taking $t = \mathsf{f}(x_1, x_2, \mathsf{a}) \in \mathbb{L}$ and $\ell = \mathsf{f}(x, x, y)$ implies $t[t|_1]_2 = \mathsf{f}(x_1, x_1, \mathsf{a}) \in \mathbb{L}$, i.e., $x_1 \in \mathcal{V}_1 \cap \mathcal{V}_2 = \varnothing$.

In the remainder of this section we assume that $\mathbb{L}$ is a layer system.

▶ **Lemma 5.6.** *Let $\mathbb{L}$ be weakly consistent with $\mathcal{R}$. If $\hat{s} \to_{p,\ell\to r} \hat{t}$ is an outer rule step then we can designate $\hat{s}$ by $\sigma(s')$ so as to find $t' \in \mathbb{L}_{\mathcal{T}}$ such that $s' \to_{p,\ell\to r} t'$ and $\hat{t} = \sigma(t')$.*

Lemma 5.6 extends Lemma 4.12 to cover outer rewrite steps. It is a key ingredient for Lemma 5.7, which shows that we can map arbitrary rewrite steps on terms to layered terms.

▶ **Lemma 5.7.** *Let $\mathbb{L}$ be weakly consistent with $\mathcal{R}$.*
1. *If $\hat{s}$ is layered and $\hat{s} \to_{\mathcal{R}} \hat{t}$ then $\hat{t}$ is layered.*
2. *If $s \to_{p,\ell\to r} t$ then $\hat{s}_{\mathbb{L}} \to_{p,\ell\to r} \cdot \rightsquigarrow \hat{t}_{\mathbb{L}}$.*

In order to prove the key Lemma 5.10, we introduce the relation $\overset{\propto}{\to}{}_i^*$. Note that because $\propto$ and $\to_i^*$ are both reflexive and transitive, so is $\overset{\propto}{\to}{}_i^*$.

▶ **Definition 5.8.** Let $\vec{t} = (t_i)_{i\in I}$ and $\vec{u} = (u_i)_{i\in I}$ be vectors. We write $\vec{t} \propto \vec{u}$ if $t_i = t_j$ implies $u_i = u_j$ for all $i, j \in I$. If $\hat{s} = \hat{C}[\![\vec{s}]\!]$, $\hat{s} \to_i^* \hat{t}$ (hence $\hat{t} = \hat{C}[\![\vec{t}]\!]$ for a suitable $\vec{t}$) and $\vec{s} \propto \vec{t}$, we write $\hat{s} \overset{\propto}{\to}{}_i^* \hat{t}$.

▶ **Lemma 5.9.** *If $\to_{\mathcal{R}}$ is confluent on layered terms of rank less than n then for layered terms of rank at most n (1) ${}_i^*\!\leftarrow \cdot \to_i^* \subseteq \overset{\propto}{\to}{}_i^* \cdot {}_i^*\!\overset{\propto}{\leftarrow}$, and (2) ${}_o^*\!\leftarrow \cdot \overset{\propto}{\to}{}_i^* \subseteq \overset{\propto}{\to}{}_i^* \cdot {}_o^*\!\leftarrow$.*

▶ **Lemma 5.10.** *Let $\mathbb{L}$ be weakly consistent with $\mathcal{R}$. If $\to_{\mathcal{R}}$ is confluent on $\mathbb{L}_{\mathcal{T}}$ and $\to_o$ is confluent on layered terms then $\to_{\mathcal{R}}$ is confluent on layered terms.*

Lemma 5.10 is proved by induction on the rank using Lemma 5.9. It is used to conclude confluence of $\to_{\mathcal{R}}$ for both our main results, Theorems 5.13 and 5.27, which we develop in Sections 5.1 and 5.2 below.

5.1 Left-Linear Systems

In this section we deal with a left-linear TRS $\mathcal{R}$ and a layer system $\mathbb{L}$ weakly consistent with $\mathcal{R}$. We want to show that $\mathcal{R}$ is confluent if $\mathcal{R}$ is confluent on $\mathbb{L}_{\mathcal{T}}$.

▶ **Lemma 5.11.** *If $\to_{\mathcal{R}}$ is confluent on $\mathbb{L}_{\mathcal{T}}$ then $\to_o$ is confluent on layered terms.*

Consequently, $\to_{\mathcal{R}}$ is confluent on layered terms by Lemma 5.10. The next lemma deals with the interaction of rewriting and fusion steps.

▶ **Lemma 5.12.** *On layered terms $\leftsquigarrow \cdot \rightsquigarrow \subseteq \rightsquigarrow \cdot \leftsquigarrow$ and $\leftsquigarrow \cdot \to_{\mathcal{R}}^* \subseteq \to_{\mathcal{R}}^* \cdot \leftsquigarrow$.*

Combining Lemmata 5.7 (to map rewrite sequences to layered terms), 5.10, 5.11 and 5.12 (for confluence of rewriting layered terms), we obtain our main result for left-linear TRSs.

▶ **Theorem 5.13.** *Let $\mathcal{R}$ be a left-linear TRS and $\mathbb{L}$ a layer system that is weakly consistent with $\mathcal{R}$. If $\mathcal{R}$ is confluent on $\mathbb{L}_{\mathcal{T}}$ then $\mathcal{R}$ is confluent.*

The above result also holds for non-duplicating TRSs, but the proof is considerably more involved. It can be found in our technical report [5].

5.2 General Systems

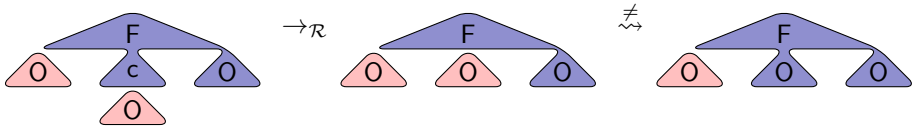
In this section we consider the case of general TRSs that may be non-left-linear. When this happens, we can conclude confluence of $\mathcal{R}$ if $\mathbb{L}$ is *consistent* with $\mathcal{R}$.

▶ **Definition 5.14.** A layer system $\mathbb{L}$ is called *consistent* with a TRS $\mathcal{R}$ if it is weakly consistent with $\mathcal{R}$ and conditions 3 and 4 hold.

3. Let $s, t \in \mathbb{L} \setminus (\mathcal{V} \cup \{\Box\})$ with $s \to_{p,\ell\to r} t$ and $q \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(\ell)$, $q' \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(r)$ such that $\ell|_q = r|_{q'}$. Furthermore let $t' \in \mathbb{L}$ be obtained from t by replacing some holes by terms from $\mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{V})$. Then $s[t'|_{pq'}]_{pq} \in \mathbb{L}$.

4. Let $s, t \in \mathbb{L}$ with $s \to_{p,\ell\to r} t$ and $q \parallel p$. If $t|_q \in \mathcal{V} \cup \{\Box\}$ and $t[t']_q \in \mathbb{L}$ then $s[t']_q \in \mathbb{L}$.

Let us examine the effect of conditions 3 and 4 of Definition 5.14. In order to deal with non-left-linear systems we have to restrict layer systems further. One key step in our proof (and the original proof in [9]) is the construction of witnesses. It is interesting to see how this fails in the counterexample from Section 3, if we define layers to be order-sorted terms, which results in a weakly consistent layer system:



Here we have a fusion step that is enabled by a rewrite step above the layer being fused. This phenomenon, *fusion from above*, defeats any attempt to build a witness in a bottom up fashion as the proof does.

The additional constraints for consistency prevent fusion from above. They demand that any subterm that can be fused with a layer *after* a rewrite step inside that layer could already have been fused *before* the rewrite step.

▶ **Example 5.15.** In the counterexample from Section 3, if we let $\mathbb{L}$ be the set of order-sorted terms closed under replacing variables by holes, we have $\mathbb{L} \ni s = \mathsf{F}(\Box, \mathsf{c}(\Box), \mathsf{O}) \to_{2,\mathsf{c}(x)\to x} \mathsf{F}(\Box, \Box, \mathsf{O}) = t \in \mathbb{L}$ and $t' = \mathsf{F}(\Box, \mathsf{O}, \mathsf{O}) \in \mathbb{L}$. With $q = 1$, $q' = \epsilon$ we conclude from Definition 5.14(3) that $s[t'|_2]_{21} = \mathsf{F}(\Box, \mathsf{c}(\mathsf{O}), \mathsf{O}) \in \mathbb{L}$. Since this term is not order-sorted, we see that $\mathbb{L}$ is not consistent with the given TRS.

As the example shows, condition 3 of Definition 5.14 needs careful consideration when the TRS $\mathcal{R}$ has collapsing rules: If $s \to_{p,\ell\to r} t$, then the subterm $s|_{pq}$ replaces $s|_p$ completely in the reduct. That is, any subterm u that can occur in place of the left-hand side of the rule application, $\sigma(\ell) = s|_p$ (meaning that $s[u]_p \in \mathbb{L}$) must also be allowed at $s|_{pq}$, i.e., $s[u]_{pq} \in \mathbb{L}$.

From now on we assume that $\mathbb{L}$ is consistent with $\mathcal{R}$.

▶ **Lemma 5.16.** *For layered terms $\hat{s}$ and $\hat{t}$, if $\hat{s} \to_o \hat{t}$ then $\hat{s}_{\mathbb{L}} \to_o \hat{t}_{\mathbb{L}}$ or both $\hat{s}_{\mathbb{L}} \to_o \epsilon(\hat{t}_{\mathbb{L}})$ and* $\mathrm{rank}(\hat{s}_{\mathbb{L}}) > \mathrm{rank}(\hat{t}_{\mathbb{L}})$.

▶ **Lemma 5.17.** *If $\to_{\mathcal{R}}$ is confluent on $\mathbb{L}_{\mathcal{T}}$ then $\to_o$ is confluent on layered terms.*

Lemma 5.17 enables us to use Lemma 5.10 again, but we still have to deal with fusion steps. The remainder of this section is based on the *simplified proof* by Klop *et al.* [9].

▶ **Definition 5.18.** A term $\hat{t}$ is called *inner preserved* if $\hat{u}|_{\_} = \hat{u}'|_{\_}$ whenever $\hat{t} \to_{\mathcal{R}}^* \hat{u} \rightsquigarrow \hat{u}'$. The term $\hat{t}$ is *preserved* if $\hat{u} = \hat{u}'$ under the same condition.

Note that the principal subterms of an inner preserved term are inner preserved as well. On inner preserved terms, fusion steps can only affect the root special position, so do not interact with rewrite steps in any essential way. Hence from Lemmata 5.17 and 5.10 we have:

▶ **Lemma 5.19.** *The relation $\rightsquigarrow_{\mathcal{R}}$ is confluent on inner preserved terms.*

▶ **Definition 5.20.** A proper fusion step $\hat{a} \overset{\neq}{\rightsquigarrow} \hat{b}$ is *inner*, denoted by $\hat{a} \to_{\ni} \hat{b}$, if $\min(P_{\hat{a}}) = \min(P_{\hat{b}})$, i.e., only non-principal positions are removed. Let $\to_{i,\ni} = \to_i \cup \to_{\ni}$. We define $\hat{s} \overset{\propto}{\to}{}^*_{i,\ni} \hat{t}$ if $\hat{C}[\![\vec{s}]\!] = \hat{s} \to_{i,\ni}^* \hat{t} = \hat{C}[\![\vec{t}]\!]$ (the top context is not affected by $\to_{i,\ni}$) and $\vec{s} \propto \vec{t}$.

The relation $\overset{\propto}{\to}{}^*_{i,\ni}$ is different from $\overset{\propto}{\to}{}^*_i$ introduced earlier in that it includes inner fusion steps. The overall intention is the same: $\overset{\propto}{\to}{}^*_{i,\ni}$ only affects principal subterms of a term and rewrites equal principal subterms in the same way.

▶ **Lemma 5.21.** *Let the relation $\rightsquigarrow_{\mathcal{R}}$ be confluent on terms of rank less than n. If $\hat{s}$ has rank n and $\hat{t}\ {}^{*}_{\Subset,i}\!\leftarrow \hat{s} \rightarrow^{*}_{i,\Supset} \hat{u}$ then $\hat{t} \overset{\propto}{\rightarrow}{}^{*}_{i,\Supset} \cdot {}^{*}_{\Subset,i}\!\overset{\propto}{\leftarrow} \hat{u}$.*

▶ **Definition 5.22.** A *witness* of a term $\hat{s}$ is an inner preserved term $\hat{t}$ such that $\hat{s} \rightsquigarrow_i^* \hat{t}$, where $\rightsquigarrow_i = \rightsquigarrow \cup \rightarrow_i$.

We can apply Lemma 5.19 to witnesses. Lemma 5.24 states that witnesses always do exist. An important step in its proof is Lemma 5.23, which allows us to replace principal subterms of a term by preserved reducts.

▶ **Lemma 5.23.** *For every inner preserved term $\hat{t}$ there exists a preserved term $\hat{u}$ with $\hat{t} \rightsquigarrow^*_{\mathcal{R}} \hat{u}$.*

▶ **Lemma 5.24.** *If $\rightsquigarrow_{\mathcal{R}}$ is confluent on terms of rank less than n then every term $\hat{t}$ with $\mathrm{rank}(\hat{t}) \leqslant n$ has a witness $\dot{t}$.*

▶ **Lemma 5.25.** *Let $\rightsquigarrow_{\mathcal{R}}$ be confluent on terms $\hat{t}$ with $\mathrm{rank}(\hat{t}) < n$. If $\hat{s} \rightsquigarrow_{\mathcal{R}} \hat{t}$ and $\mathrm{rank}(\hat{s}) = n$ then $\dot{s} \rightsquigarrow^*_{\mathcal{R}} \cdot {}^*_{\mathcal{R}}\!\rightsquigarrow \dot{t}$.*

Therefore we can construct witnesses for all terms in a conversion between two layered terms by Lemma 5.24, then join consecutive witnesses using Lemma 5.25, and finally use Lemma 5.19 to join the witnesses of the outermost terms, proving Lemma 5.26.

▶ **Lemma 5.26.** *The relation $\rightsquigarrow_{\mathcal{R}}$ is confluent if $\mathcal{R}$ is confluent on $\mathbb{L}_{\mathcal{T}}$.*

Since by Lemma 5.7 we can map rewrite sequences to layered terms, Lemma 5.26 implies our main result for arbitrary TRSs.

▶ **Theorem 5.27.** *Let $\mathcal{R}$ be a TRS and $\mathbb{L}$ a layer system that is consistent with $\mathcal{R}$. If $\mathcal{R}$ is confluent on $\mathbb{L}_{\mathcal{T}}$ then $\mathcal{R}$ is confluent.*

6 Applications

In this section we present three applications of layer systems.

Layer-Preservation: Let $\mathcal{T}_X(\mathcal{F},\mathcal{V})$ denote the set of terms with root symbol from X. Let $\mathcal{C} := \mathcal{F}_1 \cap \mathcal{F}_2$, $\mathcal{D}_1 := \mathcal{F}_1 \setminus \mathcal{F}_2$ and $\mathcal{D}_2 := \mathcal{F}_2 \setminus \mathcal{F}_1$.

▶ **Theorem 6.1** (Ohlebusch [11])**.** *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be TRSs such that $\mathcal{R}_1 \subseteq \mathcal{T}(\mathcal{C},\mathcal{V})^2 \cup \mathcal{T}_{\mathcal{D}_1}(\mathcal{F}_1,\mathcal{V})^2$, $\mathcal{R}_2 \subseteq \mathcal{T}(\mathcal{C},\mathcal{V})^2 \cup \mathcal{T}_{\mathcal{D}_2}(\mathcal{F}_2,\mathcal{V})^2$ and $\mathcal{R}_1 \cap \mathcal{T}(\mathcal{C},\mathcal{V})^2 = \mathcal{R}_2 \cap \mathcal{T}(\mathcal{C},\mathcal{V})^2$. The union $\mathcal{R}_1 \cup \mathcal{R}_2$ is confluent if and only if $\mathcal{R}_1$ and $\mathcal{R}_2$ are confluent.*

Proof Sketch. Let $\mathbb{L} := \mathcal{T}(\mathcal{C} \cup \{\Box\},\mathcal{V}) \cup \mathcal{T}_{\mathcal{D}_1}(\mathcal{F}_1 \cup \{\Box\},\mathcal{V}) \cup \mathcal{T}_{\mathcal{D}_2}(\mathcal{F}_2 \cup \{\Box\},\mathcal{V})$. It is straightforward (but somewhat tedious) to verify that with the given constraints, this is a layer system that is consistent with $\mathcal{R}_1 \cup \mathcal{R}_2$. Confluence follows by Theorem 5.27. ◀

Order-Sorted Persistence: Note that many-sorted persistence [1] arises as a special case of Theorem 6.2 by making sorts mutually incomparable, which in turn entails Toyama's classical modularity result [13]. The conditions are easy to implement, as outlined in [15].

▶ **Theorem 6.2.** *Let $\mathcal{R}$ be an $(\mathcal{S},\succeq)$-order-sorted TRS. Assume the following conditions:*

1. *$\mathcal{R}$ is compatible with $\mathcal{S}$, i.e., for $\ell \to r \in \mathcal{R}$ the terms ℓ and r are order-sorted with variable occurrences strictly bound in ℓ and $\mathrm{sort}(\ell) \succeq \mathrm{sort}(r)$.*

2. *If $\mathcal{R}$ is non-left-linear then for $\ell \to r \in \mathcal{R}$, variable occurrences in r are strictly bound as well. Furthermore, for collapsing rules ($r \in \mathcal{V}$) the sort of r must be maximal.*

If $\mathcal{R}$ is confluent on order-sorted terms then $\mathcal{R}$ is confluent on all terms.

Proof Sketch. The proof idea is to use $\mathbb{L} = \mathcal{T}_{\mathcal{S}}(\mathcal{F}, \mathcal{V})$ as a layer system, after closing it under replacing variables by holes. Conditions (1) and (2) of Theorem 6.2 ensure weak consistency and consistency of $\mathbb{L}$ with $\mathcal{R}$, respectively. Condition (1) also makes $\mathbb{L}$ closed under (unsorted) rewriting, by only allowing variables x to be assigned terms t having $\mathrm{sort}(t) \preceq \mathrm{sort}(x)$ when matching rules. Theorems 5.13 and 5.27 conclude the proof. ◀

Actually, condition (2) in the above theorem can be weakened to non-left-linear and duplicating TRSs, see [5].

There are a couple of notable differences between Theorem 6.2 and [1, Theorem 4.12]. First, the maximality constraint on the sorts of collapsing rules in the general case is new, and indeed rules out the counterexample from Section 3. Second, we have weaker constraints on left-linear systems. Finally, we do not require the order on sorts to be well-founded. (This is needed in [1] because every step that weakens the sort of a special subterm is treated as destructive, while in our approach only actual fusion steps must be considered.)

Theorem 6.2 is strictly stronger than many-sorted persistence, as the next example shows.

▶ **Example 6.3** (adapted from [1]). Consider the TRS $\mathcal{R}$ consisting of the rewrite rules

$$1\colon \mathsf{f}(x,\mathsf{A}) \to \mathsf{G}(x) \qquad 2\colon \mathsf{f}(x,\mathsf{f}(x,\mathsf{B})) \to \mathsf{B} \qquad 3\colon \mathsf{G}(\mathsf{C}) \to \mathsf{C} \qquad 4\colon \mathsf{F}(x) \to \mathsf{F}(\mathsf{G}(x))$$

and the sorts $\mathcal{S} = \{0,1,2\}$ with $1 \succeq 0$ and the signature given by $\mathsf{A},\mathsf{B} : 1$, $\mathsf{C} : 0$, $\mathsf{F} : 0 \to 2$, $\mathsf{G} : 0 \to 0$, $\mathsf{f} : 0 \times 1 \to 1$ and $x : 0$. It is straightforward to check that the requirements of Theorem 6.2 are fulfilled. In the order-sorted TRS, only rules (1), (2) and (3) can be applied to terms of sort 1 and their derivatives, rules (3) and (4) can be applied to terms derived from terms of sort 2 and only rule (3) can be applied to terms of sort 0. Hence, since $\mathcal{R}_1 = \{(1),(2),(3)\}$ (which is terminating and has no critical pairs), $\mathcal{R}_2 = \{(3),(4)\}$ (which is orthogonal), and $\mathcal{R}_3 = \{(3)\}$ (orthogonal) are confluent, $\mathcal{R}$ is confluent. No such decomposition can be obtained with many-sorted persistence. Consider a *most general* signature making all rules many-sorted: $\mathsf{A},\mathsf{B},\mathsf{C},x : 0$, $\mathsf{F} : 0 \to 1$, $\mathsf{G} : 0 \to 0$, and $\mathsf{f} : 0 \times 0 \to 0$. Since terms of sort 1 can have subterms of sort 0, no decomposition is possible.

The weaker conditions in Theorem 6.2 for left-linear TRSs are beneficial.

▶ **Example 6.4.** Consider the TRS $\mathcal{R}$ consisting of the rewrite rules

$$\mathsf{c}(x) \to x \qquad \mathsf{f}(\mathsf{A}) \to \mathsf{f}(\mathsf{f}(\mathsf{c}(0))) \qquad \mathsf{g}(\mathsf{B}) \to \mathsf{g}(\mathsf{g}(\mathsf{c}(0)))$$

the sorts $\mathcal{S} = \{0,1,2\}$ with $1,2 \succeq 0$ and the signature given by $0, x : 0$, $\mathsf{c} : 0 \to 0$, $\mathsf{A} : 1$, $\mathsf{f} : 1 \to 1$, $\mathsf{B} : 2$ and $\mathsf{g} : 2 \to 2$. This satisfies the conditions of Theorem 6.2 for left-linear systems, and we can decompose the system into the component induced by sort 1: $\{\mathsf{c}(x) \to x, \mathsf{f}(\mathsf{A}) \to \mathsf{f}(\mathsf{f}(\mathsf{c}(0)))\}$ and sort 2: $\{\mathsf{c}(x) \to x, \mathsf{g}(\mathsf{B}) \to \mathsf{g}(\mathsf{g}(\mathsf{c}(0)))\}$. If we add the restrictions for non-left-linear systems, the collapsing rule $\mathsf{c}(x) \to x$ enforces $\mathsf{c} : \alpha \to \alpha$ for a maximal sort α. Hence also the argument of f and g has sort α, and $\alpha \succeq \mathrm{sort}(\mathsf{A}), \mathrm{sort}(\mathsf{B}), \mathrm{sort}(\mathsf{f}(x)), \mathrm{sort}(\mathsf{g}(x)), \mathrm{sort}(0)$. So the component induced by α contains all rules.

Currying: Currying is a transformation of TRSs that introduces partial applications. It is useful in the construction of polynomial-time procedures for deciding properties of TRSs as, for example, in [4]. Kahrs [7] proved that confluence is preserved by this transformation.

▶ **Definition 6.5.** Let $\mathcal{R}$ be a TRS over a signature $\langle \mathcal{F}, \mathcal{V} \rangle$. Let $\mathcal{F}' = \mathcal{F} \cup \{\mathsf{Ap}\}$ where Ap is a binary function symbol and all function symbols in $\mathcal{F}$ become constants. The *curried version* of $\mathcal{R}$ operates on $\mathcal{T}(\mathcal{F}', \mathcal{V})$ and is defined as $\mathsf{Cu}(\mathcal{R}) = \{\mathsf{Cu}(\ell) \to \mathsf{Cu}(r) \mid \ell \to r \in \mathcal{R}\}$. Here $\mathsf{Cu}(t) = t$ if t is a variable or a constant and $\mathsf{Cu}(f(t_1, \ldots, t_n)) = \mathsf{Ap}(\cdots \mathsf{Ap}(f, \mathsf{Cu}(t_1)) \cdots, \mathsf{Cu}(t_n))$ (with n occurrences of Ap). Let $\mathcal{F}'' = \mathcal{F}' \cup \{f_i \mid f \in \mathcal{F}, 0 \leqslant i < \mathrm{arity}(f)\}$, where f_i has arity i. The *partial parametrization* $\mathsf{PP}(\mathcal{R})$ of $\mathcal{R}$ is defined as the union of $\mathcal{R}$ and $\mathcal{U}$, where $\mathcal{U}$ consists of all *uncurrying* rules:

$$\mathsf{Ap}(f_i(x_1, \ldots, x_i), x_{i+1}) \to f_{i+1}(x_1, \ldots, x_{i+1})$$

for all $f \in \mathcal{F}$ and $0 \leqslant i < \mathrm{arity}(f)$, with the convention that $f_n = f$ if $n = \mathrm{arity}(f)$.

Note that $\to_{\mathcal{U}}$ is terminating and confluent (because $\mathcal{U}$ is orthogonal). Partial parametrization is closely related to currying: by [7, Proposition 3.1] and [8, Theorem 2.2], $\mathsf{PP}(\mathcal{R})$ is confluent if and only if $\mathsf{Cu}(\mathcal{R})$ is confluent.

▶ **Theorem 6.6.** *If $\mathcal{R}$ is confluent then $\mathsf{PP}(\mathcal{R})$ is confluent.*

Proof Sketch. Consider the term $t = \mathsf{Ap}(\cdots \mathsf{Ap}(f_i(t_1, \ldots, t_i), t_{i+1}) \cdots, t_m)$ where $f \in \mathcal{F}$, which is a curried function application of f with m arguments. Let t'_i be the $\to_{\mathcal{U}}$ normal form of t_i and $n = \mathrm{arity}(f)$. If $m < n$, the $\to_{\mathcal{U}}$ normal form of t will be $f_m(t'_1, \ldots, t'_m)$, and we call the function application *partial*. Otherwise, $m \geq n$, and the $\to_{\mathcal{U}}$ normal form of t is $\mathsf{Ap}(\cdots \mathsf{Ap}(f(t'_1, \ldots, t'_n), t'_{n+1}) \cdots, t'_m)$. We call $\mathsf{Ap}(\cdots \mathsf{Ap}(f_i(t_1, \ldots, t_i), t_{i+1}) \cdots, t_n)$ a *saturated* function application, and t_{n+1} to t_m *extra* arguments of f. We define $\mathbb{L} = \mathbb{L}_1 \cup \mathbb{L}_2 \cup \mathbb{L}_3$ where

- $\mathbb{L}_1 = \{t \mid t \to^*_{\mathcal{U}} u \text{ with } u \in \mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{V})\}$,
- $\mathbb{L}_2 = \{t \mid t \to^*_{\mathcal{U}} f_n(x_1, \ldots, x_n),\ f \in \mathcal{F},\ x_i \in \mathcal{V} \cup \{\Box\},\ n < \mathrm{arity}(f)\}$,
- $\mathbb{L}_3 = \{\mathsf{Ap}(x_1, x_2) \mid x_i \in \mathcal{V} \cup \{\Box\}\}$.

The idea is to start a new layer for any partial function application ($\mathbb{L}_2$), any Ap that binds extra arguments and thus remains in the $\to_{\mathcal{U}}$ normal form of t ($\mathbb{L}_3$), and for each extra argument, but not for any saturated function applications that appear as arguments of other saturated function applications ($\mathbb{L}_1$). It can be shown that $\mathbb{L}$ is indeed a layer system that is consistent with $\mathcal{R}$. ◀

7 Conclusion

In this paper we have presented an abstract layer framework that covers several known results about the modularity and persistence of confluence. The framework enabled us to correct the result claimed in [1] on order-sorted persistence, and, by weaker conditions for left-linear systems, to increase its applicability. We also showed how Kahrs' confluence result for curried systems is obtained as an instance of our layer framework. Furthermore, we have incorporated a decomposition technique due to Theorem 6.2 into CSI [15], our confluence prover.

As future work, we plan to investigate how to apply layer systems to other properties of TRSs, like termination or having unique normal forms. Since the applications in this paper use regular languages for the layer system $\mathbb{L}$, we also plan to investigate how the consistency conditions translate into restrictions on tree automata. Another interesting

question is whether van Oostrom's constructive modularity proof [14] can be adapted for layer systems. Finally, we worked out the technical details of our main results to prepare for future certification efforts in a theorem prover like Isabelle.

Acknowledgments

We thank the anonymous reviewers for their helpful and detailed comments.

References

1 T. Aoto and Y. Toyama. Extending persistency of confluence with ordered sorts. Technical Report IS-RR-96-0025F, School of Information Science, JAIST, 1996.

2 T. Aoto and Y. Toyama. Persistency of confluence. *Journal of Universal Computer Science*, 3(11):1134–1147, 1997.

3 F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

4 H. Comon, G. Godoy, and R. Nieuwenhuis. The confluence of ground term rewrite systems is decidable in polynomial time. In *Proc. 42nd Annual Symposium on Foundations of Computer Science*, pages 298–307, 2001.

5 B. Felgenhauer, H. Zankl, and A. Middeldorp. Layer systems for proving confluence. Technical report, University of Innsbruck, 2011. Available at `http://cl-informatik.uibk.ac.at/software/csi/layerframework_report.pdf`.

6 J.-P. Jouannaud and Y. Toyama. Modular Church-Rosser modulo: The complete picture. *International Journal of Software and Informatics*, 2(1):61–75, 2008.

7 S. Kahrs. Confluence of curried term-rewriting systems. *Journal of Symbolic Computation*, 19(6):601–623, 1995.

8 R. Kennaway, J.W. Klop, M. Ronan Sleep, and F.-J. de Vries. Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*, 21(1):15–39, 1996.

9 J.W. Klop, A. Middeldorp, Y. Toyama, and R. de Vrijer. Modularity of confluence: A simplified proof. *Information Processing Letters*, 49:101–109, 1994.

10 C. Lüth. Compositional term rewriting: An algebraic proof of Toyama's theorem. In *Proc. 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 261–275, 1996.

11 E. Ohlebusch. *Modular Properties of Composable Term Rewriting Systems*. PhD thesis, Universität Bielefeld, 1994.

12 B.K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973.

13 Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.

14 V. van Oostrom. Modularity of confluence constructed. In *Proc. 4th International Joint Conference on Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, pages 348–363, 2008.

15 H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. 23rd International Conference on Automated Deduction*, volume 6803 of *Lecture Notes in Artificial Intelligence*, pages 499–505, 2011.

16 H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, 1994.

The Semi-stochastic Ski-rental Problem

Aleksander Mądry[1] and Debmalya Panigrahi[2]

1 **Microsoft Research New England**
Cambridge, MA, USA
madry@mit.edu

2 **Computer Science and Artificial Intelligence Laboratory**
Massachusetts Institute of Technology
Cambridge, MA, USA
debmalya@mit.edu

Abstract

In this paper, we introduce the *semi-stochastic* model for dealing with input uncertainty in optimization problems. This model is a hybrid between the overly pessimistic online model and the highly optimistic stochastic (or Bayesian) model. In this model, the algorithm can obtain only *limited* stochastic information about the future (i.e. about the input distribution)—as the amount of stochastic information we make available to the algorithm grows from no information to full information, we interpolate between the online and stochastic models. The central question in this framework is the trade-off between the performance of an algorithm, and the stochastic information that it can access. As a first step towards understanding this trade-off, we consider the *ski-rental* problem in the semi-stochastic setting. More precisely, given a desired competitive ratio, we give upper and lower bounds on the amount of stochastic information required by a deterministic algorithm for the ski-rental problem to achieve that competitive ratio.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases online optimization, stochastic algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.300

1 Introduction

In many real-world optimization problems, the input to the problem is not known at the outset, but is revealed slowly during the execution of the optimization algorithm. For example, in the *facility location* problem, where the goal is to select a minimum-cost set of locations for setting up facilities that can serve all clients, the exact set of clients is often not known in advance. Traditionally, such problems have been modeled by the *online* framework, where the algorithm is required to satisfy a (growing) set of constraints, while attempting to optimize the objective function. For example, in the facility location problem, the algorithm will need to ensure that at any stage, the facilities set up can serve the client requests received till that stage, while minimizing the cost of facilities. The performance of online algorithms is usually measured using the notion of *competitive ratio*, which is the maximum — taken over all the possible input sequences — of the ratio of the objective function in the solution produced by the algorithm to that of the optimal offline solution (i.e. the optimal solution when the entire input is known in advance).

After an initial period of vibrant research activity in online algorithms (see [1] for a survey), there has been a slight lull in recent years, partly due to the inherent pessimism of the online model, which rendered a lot of natural optimization tasks unnecessarily hard. For example, in the facility location problem, while the exact composition of the clientele is typically not

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 300–311

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

known in advance, a rough estimate of the clientele distribution is possible to obtain from e.g. market surveys. This has motivated the development of the *stochastic optimization* model (sometimes called the *Bayesian* model) in recent times, where the probability distribution over possible input sequences is known in advance, though the exact input sequence is unknown. The performance of such online algorithms is typically measured using one of the following two metrics:

- **Expectation of ratio (EoR).** The expected value of the competitive ratio of the algorithm when the input is drawn from the worst-case distribution.
- **Ratio of expectation (RoE).** The ratio of the expected value of the objective function in the algorithmic solution to the expected value of the objective function in the offline optimal solution, when the input is drawn from the worst-case distribution.

In recent years, algorithms have been developed for stochastic versions of various fundamental optimization problems (see related work for some examples). However, in many situations, the stochastic optimization setting is overly optimistic. For example, in the facility location problem described above, while it is possible to obtain a rough estimate of the clientele distribution, such information typically comes from market surveys which are expensive to conduct. In particular, to obtain fine-grained information, one needs to conduct surveys using a large sample, thereby incurring significant cost. In essence, there is a cost associated with obtaining stochastic information about the input for an online optimization problem, and this cost grows as the algorithm seeks more detailed information about the distribution from which the input is drawn. In this correspondence, we take a step toward understanding this trade-off between the performance of an online algorithm and the cost it needs to pay to obtain stochastic information about the input.

We propose a model for stochastic optimization of online problems based on the following two-stage game between the *algorithm* and the *adversary*. In the first stage, which comprises multiple rounds, the algorithm gathers stochastic information from the adversary about the distribution of the input. Each round comprises a *query* asked by the algorithm about the input distribution that the adversary answers. The algorithm is *adaptive*, i.e. it can choose its next query based on the answers given by the adversary to its previous queries. However, the maximum number of queries that the algorithm can ask is given in advance — we call this the *query budget* of the algorithm. The first stage ends when the algorithm has exhausted its query budget. The second stage comprises an instance of the online problem where the adversary now presents the actual input sequence, which must be drawn from some distribution that is consistent with the stochastic information provided during the first stage. This allows the algorithm to use the information it acquired in the first stage to guide its choices. We call this the *semi-stochastic* model for optimization problems.

As in stochastic optimization, the performance of an algorithm in the semi-stochastic model can be measured using either the EoR or the RoE metrics. However, we need to be careful about how we define a *worst-case* distribution. The performance of the algorithm for a fixed distribution over the input space is defined as its competitive ratio for the worst-case sequence of (truthful) answers given by the adversary to the queries asked by the algorithm in the first stage. In other words, given a fixed distribution over the inputs that the adversary knows but the algorithm does not, one can visualize the first stage as a game where the algorithm aims to minimize its expected competitive ratio (which is measured using either the EoR or the RoE metric) and the adversary aims to maximize it. The overall performance of the algorithm is the worst-case performance over all distributions over the input space, where the performance for a fixed distribution is as described above.

The semi-stochastic model may be viewed as a hybrid between the pessimistic online

model and the optimistic stochastic model. In particular, if the algorithm has zero query budget, then we obtain the online model, while if the query budget is infinite, then we get the stochastic model. The goal of introducing this model is to study the trade-off between the query budget given to an online algorithm and its performance. As described earlier, a large query budget typically implies a large cost of obtaining such fine-grained information about the input distribution and might not be useful unless it improves the performance of the algorithm significantly.

It is instructive at this point to compare the semi-stochastic model to the well-known computational learning paradigm. The goal in computational learning is to learn a *hypothesis class* (i.e. a subspace of some fixed space) given the labeling of a few sample points from this space (i.e. whether each point is in the subspace or not). Typically, one is interested in the trade-off between the size of the sample and the accuracy of the guessed hypothesis. At a high level, one may interpret the semi-stochastic model as an extension of the computational learning paradigm to optimization problems. Here, our goal is to *learn* the input distribution to an optimization problem for the purpose of designing an algorithm with small competitive ratio while minimizing the amount of information about the distribution that we explicitly ask for in the form of queries.

It is important to note that we have not specified the exact nature of queries that the adversary is allowed to ask in the first stage. In principle, a particular instantiation of our model might allow for extremely powerful queries thereby letting a semi-stochastic algorithm gain more information about the input distribution than a corresponding stochastic algorithm that has access to the input distribution via a computationally bounded process, e.g. a polynomial number of samples. However, we expect that in most interesting instantiations of the semi-stochastic model, including the one that we will consider in this paper, the algorithm will be limited to some class of *computationally bounded queries*, i.e. queries that can be answered (to the desired accuracy) using polynomial amount of information regarding the input distribution.

1.1 An application: The ski-rental problem

Perhaps the most fundamental of online problems, the *ski-rental* problem is defined as follows. Suppose one wants to go skiing, but does not know the exact number of days that she would like to ski. Given this uncertainty, she needs to decide every morning whether she would like to buy a pair of skis that cost b for the entire season, or rent skis for the day paying 1. Overall, she would like to minimize her expenditure. The ski-rental problem (or its slight variants) have been used to solve various online problems, e.g. TCP acknowledgement, Bahncard problem, etc [14]. It is well-known that the ski-rental problem has a simple 2-competitive deterministic algorithm, and this is optimal for deterministic algorithms. Karlin *et al* [15] showed that the competitive ratio can be improved to $\frac{e}{e-1}$ in expectation, if the online algorithm is allowed to be randomized, and that this is optimal for randomized algorithms.

In this paper, we analyze the ski-rental problem in the semi-stochastic model as a first step towards investigating more complicated online problems. First, we note that if we employ the RoE performance metric in this problem, then we encounter the following difficulty. The adversary can create a distribution where an arbitrarily small probability mass is at $x \neq 0$ and all the remaining probability mass is at 0. (Unless otherwise stated, x will represent the duration of skiing which is the only input parameter for the ski-rental problem.) Then, in any reasonable query model, the algorithm will require an unbounded query budget to achieve any RoE value less than the trivial 2. We therefore focus on the EoR metric; in the remainder of the paper, the competitive ratio of an algorithm in the semi-stochastic model

will refer to the EoR metric. Further, as is standard in the literature (see e.g. [14]), we consider a continuous version of this problem. This allows us, in particular, to scale the price of the skis and assume wlog that $b = 1$—we adopt this convention from now on. The input is represented by a probability density function $p(x)$ such that $\int_0^\infty p(x)d(x) = 1$. Any deterministic strategy for this problem is parameterized by the value $k \geq 0$ such that skis are rented as long as $x < k$ and bought at $x = k$. We denote this strategy by $S(k)$, and its competitive ratio is given by

$$r(k) := \begin{cases} \int_0^k p(x)dx + (1+k)\int_k^\infty \frac{p(x)}{\min\{x,1\}}dx & \text{if } k \leq 1 \\ \int_0^1 p(x)dx + \int_1^k xp(x)dx + (1+k)\int_k^\infty p(x)dx & \text{if } k > 1. \end{cases}$$

Before proceeding further, we need to define our query model, i.e. the allowed set of queries that the algorithm can ask the adversary in the first stage. We consider the setting where each query is a quantile query, i.e. the algorithm gives a value $0 < y < 1$, and the adversary in response gives a number ℓ such that $\int_0^\ell p(x)dx = y$. Note that after the algorithm has asked a set of q queries, it can partition $[0, \infty)$ into a set of $q+1$ non-overlapping intervals such that the probability mass in each interval is known but the distribution within an interval is unknown to the algorithm. If the algorithm asks a large number of queries, then these intervals are fine-grained and therefore the algorithm has detailed information about the input distribution whereas if its query budget is small, then it obtains a very coarse-grained description of this distribution.

Note that the existence of the $\frac{e}{e-1}$-competitive randomized algorithm implies that for any input distribution, there exists a deterministic strategy $S(k)$ that achieves a competitive ratio of $\frac{e}{e-1}$. If not, then the corresponding distribution can be used with Yao's minimax principle [23] to prove a randomized lower bound greater than $\frac{e}{e-1}$ contradicting the algorithm of Karlin *et al.* Our goal, in this paper, is to determine the minimum query budget of a deterministic semi-stochastic algorithm that has a competitive ratio of $\frac{e}{e-1} + \epsilon$ for any $\epsilon > 0$. (Of course, we are interested in small fractions ϵ, since there is an online deterministic strategy that has a competitive ratio of 2.) To this end, we prove the two following theorems.

▶ **Theorem 1** (Lower Bound). *There exists a universal constant $C > 0$ such that for any constant $\varepsilon \in \left(0, \frac{e-2}{4(e-1)}\right]$,[1] if the algorithm has a quota of $q < \frac{C}{\varepsilon}$ queries in the first stage, then for any deterministic strategy $S(k)$ chosen in the second stage, there exists an input distribution $\mathcal{D}$ that has the following properties:*

- *The distribution $\mathcal{D}$ has the same quantiles as those provided by the adversary in the first stage.*
- *The expected competitive ratio of the algorithm $S(k)$ for input distribution $\mathcal{D}$ is greater than $\frac{e}{e-1} + \varepsilon$.*

▶ **Theorem 2** (Upper Bound). *For any $\epsilon > 0$, there is a polynomial-time deterministic algorithm with query budget $O\left(\frac{1}{\epsilon^{3/2}}\right)$ that has a competitive ratio of $\frac{e}{e-1} + \epsilon$.*

We find the lower bound somewhat surprising. Note that for most input distributions, an algorithm needs to know only some parts of the distribution in detail, but can do with more coarse-grained information about the other parts. Therefore, it is conceivable that an

[1] The range of ε in the theorem can be increased from $\left[0, \frac{e-2}{4(e-1)}\right]$ to $\left[0, \frac{\alpha(e-2)}{e-1}\right]$ for any $\alpha < 1$ with the only change being in the value of the constant C. At $\alpha = 1$, the lower bound must fail since there exists a 2-competitive deterministic online algorithm. Thus, $\lim_{\alpha \to 0} C = \infty$. Since we are primarily interested in the asymptotic behavior of the lower bound for small ε, and to maintain simplicity of notation, we consider the representative value of $\alpha = 1/4$ in the rest of the paper.

adaptive algorithm recognizes which parts of the distribution it needs to query in greater detail and therefore makes do with less than polynomial in $1/\epsilon$ queries overall. However, our lower bound proves that even when the algorithm is adaptive, it still requires a large (polynomial in $1/\varepsilon$) number of queries to pinpoint — up to an accuracy of ε — the parts of the input distribution that are more critical.

1.2 Our Techniques

Our algorithm for proving the upper bound obtains a set of equi-spaced quantiles in the first stage by asking the corresponding queries. Once it has obtained these quantiles, it assumes an arbitrary density function on each of the inter-quantile intervals, and computes the optimal deterministic algorithm for this assumed input distribution. We would then need to claim that no matter how much the actual density function in each inter-quantile interval differs from the assumed one, the performance of the algorithm does not degrade significantly. It turns out that for most natural choices of the assumed density function in each interval, this claim does not hold, i.e. the adversary can change the input distribution while not changing any of the quantiles to degrade the competitive ratio of the algorithm significantly. However, our crucial observation is that the claim would hold if the value of k in the deterministic strategy is large enough, i.e. the adversary can significantly degrade the performance of the algorithm only if it selects a small value of k. To make the algorithm robust to this possibility, our algorithm deliberately chooses a large enough value of k that might be sub-optimal for the assumed distribution. To prove that the sub-optimality arising from this choice of k can be absorbed in our competitive ratio, we show for any input distribution, there always exists a large enough value of k that is *almost* optimal.

Interestingly, our proof of the lower bound is somewhat related to the problem we highlighted above. Namely, it is established by making the adversary answer all the queries of the algorithm according to a certain distribution that penalizes strategies that buy skis later, forcing the algorithm to choose a strategy that buys relatively early. Next, we show that in this case, unless the algorithm is able to pinpoint the distribution beyond k (where the algorithm's chosen strategy is $S(k)$) up to accuracy $O(\varepsilon)$, the adversary can always modify the distribution to make the algorithm suffer a large competitive ratio, while retaining compatibility with the stochastic information provided earlier to the algorithm.

1.3 Related Work

The term *stochastic optimization* has been used to mean various algorithmic models in the literature. As described above, we use it to imply that the algorithm knows the distribution of the input but not the input itself. Various optimization problems have been studied in this model previously, including facility location [19], network design problems [20, 9], secretary-type problems [2, 17, 16, 18], matching [10, 8, 5, 3], packing and covering problems [6, 7, 11], etc. Another closely related model where significant progress has been reported in recent years is two-stage stochastic optimization with recourse (see [21] for a survey). Here, the algorithm has a choice of buying resources at a lower cost with only stochastic knowledge of the input, or at a higher cost with knowledge of the actual input. Various other models that lie between the online model and the stochastic model have also been considered in the literature, especially for online problems with multiple inputs. Among these, the two most important models are perhaps the i.i.d. from unknown distribution model where each input is drawn independently and identically from a distribution that is not known to the algorithm, and the random permutation model where the input sequence is a random permutation on

an adversarially chosen input set. In the ski-rental problem, there is only one input, and therefore, these multi-input models are not relevant.

A class of problems that have some similarity with the questions that we are exploring are multi-armed bandit (MAB) problems (see e.g. [4]). These problems model the classical "exploration-vs-exploitation" trade-off in decision problems. The particular variant of this problem that is most similar to our problem is the *budgeted learning* problem [12, 13], where a budgeted exploration phase is followed by an exploitation phase. However, the typical setup in these problems is that one has to choose from multiple options, each of which has an unknown associated reward, and the algorithm needs to explore the options within a stipulated budget so that it can maximize revenue (or minimize regret) later in the exploitation phase. On the other hand, our model pertains to optimization problems, and the goal of the exploration phase is to gain sufficient knowledge about the input space for optimizing the objective function in the exploitation phase.

1.4 Roadmap

In section 2, we prove the lower bound on the query complexity of the ski-rental problem in the semi-stochastic model (Theorem 1); the corresponding upper bound (Theorem 2) appears in section 3. Finally, in section 4, we give some directions for future work on the semi-stochastic model.

2 A Lower Bound on the Competitive Ratio of Semi-Stochastic Algorithms for the Ski-rental Problem

In this section, our goal is to prove Theorem 1, i.e. to give a lower bound on the number of queries that *any* semi-stochastic algorithm needs to ask in order to have an expected competitive ratio of at most $\frac{e}{e-1}+\epsilon$. As earlier, we will assume wlog that the cost of buying skis is 1, and $S(k)$ will represent the deterministic strategy that rents skis until the duration of skiing x reaches k, at which point skis are bought.

Consider a probability distribution $\mathcal{D}_0$ on the input given by the following density function for $0 \leq x \leq 1$: $\mathcal{D}_0(x) = \left(\frac{e}{e-1}\right) xe^{-x}$. In addition, we have a probability mass of $\frac{1}{e-1}$ at $x = +\infty$.[2] For any $0 \leq \delta < 1$, consider the distribution $\mathcal{D}_\delta$ created by scaling the density function of $\mathcal{D}_0$ by $1-\delta$ in the range $0 \leq x \leq 1$, and shifting the surplus probability mass of $\delta\left(\frac{e-2}{e-1}\right)$ to $+\infty$. The density function for this distribution in the range $0 \leq x \leq 1$ is given by $\mathcal{D}_\delta(x) = (1-\delta)\left(\frac{e}{e-1}\right) xe^{-x}$. In addition, there is a probability mass of $\frac{1+(e-2)\delta}{e-1}$ at $x = +\infty$. The next lemma lower bounds the expected competitive ratio of deterministic strategies for distribution $\mathcal{D}_\delta$.

▶ **Lemma 3.** *For any $k \geq 0$ and $0 \leq \delta < 1$, the expected competitive ratio of strategy $S(k)$ where the input has distribution $\mathcal{D}_\delta$ is $\frac{e}{e-1} + \delta\left(k - \frac{1}{e-1}\right)$ if $0 \leq k \leq 1$ and at least $\frac{e}{e-1} + \delta k(1 - \frac{1}{e-1})$ if $k > 1$.*

[2] We can use Dirac delta function [22] to define the density function at $x = +\infty$ in a mathematically precise manner, but we will ignore this technicality throughout the paper and assume a probability mass of $\frac{1}{e-1}$ at $x = +\infty$.

Proof. For $0 \leq k \leq 1$, the expected competitive ratio of strategy $S(k)$ is

$$\frac{e(1-\delta)}{e-1}\left(\int_0^k xe^{-x}dx + \int_k^1 (1+k)e^{-x}dx\right) + \left(\frac{1+k}{e-1}\right)(1+\delta(e-2)) = \frac{e}{e-1} + \delta\left(k - \frac{1}{e-1}\right).$$

On the other hand, if $k > 1$, then the expected competitive ratio of strategy $S(k)$ is

$$\int_0^1 \frac{e(1-\delta)}{e-1}xe^{-x}dx + \left(\frac{1+k}{e-1}\right)(1+\delta(e-2)) \geq \frac{e}{e-1} + \delta k\left(1 - \frac{1}{e-1}\right). \quad \blacktriangleleft$$

This lemma implies that as δ grows, the distribution $\mathcal{D}_\delta$ favors strategies $S(k)$ for small values of k by making their expected competitive ratio less than $\frac{e}{e-1}$, while making the expected competitive ratio of strategies $S(k)$ with large values of k worse. This property of distribution $\mathcal{D}_\delta$ will be crucial in our lower bound construction.

We are now ready to prove our lower bound. We will use the following fact.

▶ Fact 1. For any $t \geq 0$ and $0 \leq \Delta \leq 1$, $\int_t^{t+\Delta}\left(\frac{x}{t} - 1\right)e^{-x}dx \geq \frac{e^{-t}}{t}\Delta^2$.

Proof. We have

$$\int_t^{t+\Delta}\left(\frac{x}{t} - 1\right)e^{-x}dx = \frac{e^{-t}}{t}\left(1 - e^{-\Delta}(1+\Delta)\right) \geq \frac{e^{-t}}{t}\left(1 - (1-\Delta)(1+\Delta)\right) = \frac{e^{-t}}{t}\Delta^2. \quad \blacktriangleleft$$

Proof of Theorem 1. In the first round, for any query asked by the algorithm, the adversary returns the corresponding quantile of the distribution $\mathcal{D}_\delta$ for some δ whose value (that will depend only on ε) we will fix later. This partitions the entire input interval $[0, \infty)$ into contiguous non-overlapping intervals with the quantiles representing the boundaries between adjacent intervals. Let these quantiles be $y_0 = 0, y_1, \ldots, y_q$, where $y_i \leq y_{i+1}$ for each i. Note that for each interval $[y_i, y_{i+1}]$, the algorithm knows the probability that the input is contained in it, though it does not know the exact distribution.

For notational convenience, we assume wlog that the strategy $S(k)$ chosen by the algorithm satisfies $k = y_s$ for some $0 \leq s \leq q$.[3] Based on the values of y_is and the choice of k, the adversary chooses the actual distribution $\mathcal{D}$ of the input. This distribution is obtained from $\mathcal{D}_\delta$ by concentrating the entire probability mass in each interval (y_i, y_{i+1}) where $i \geq s$, at a value y_i^+ that is infinitesimally close to (but greater than) y_i, while leaving the probability distribution of $x < y_s$ and that of $x = +\infty$ unchanged. Formally, the input distribution $\mathcal{D}$ is given by the following density function, where y_{q+1} is any finite value that is greater than $\max\{1, y_q\}$:

$$\mathcal{D}(x) = \begin{cases} \int_{y_i^+}^{y_{i+1}} \mathcal{D}_\delta(x)dx & \text{for each } x = y_i^+, \text{ where } s \leq i \leq q \\ \mathcal{D}_\delta(x) & \text{if } x \leq y_s \text{ or } x = +\infty \\ 0 & \text{otherwise.} \end{cases}$$

Note that by construction, the quantiles $y_1, y_2, \ldots, y_q$ hold for distribution $\mathcal{D}$ as well.

First, note that the expected competitive ratio of strategy $S(y_s)$ for distribution $\mathcal{D}_\delta$ is at most as much as that for distribution $\mathcal{D}$. Further, the difference between these expected

[3] If the algorithm chooses $y_s < k < y_{s+1}$ for some s, then the adversary uses the same distribution as that for $k = y_s$ except that the probability mass in the interval (y_s, y_{s+1}) is concentrated at x^+ rather than at y_s^+. Clearly, the expected competitive ratio of the algorithm for this input distribution is worse than the expected competitive ratio for the corresponding input distribution if the algorithm had chosen $k = y_s$.

competitive ratios is only due to the difference in the probability density function in the range $y_s < x \leq 1$. Let y_r be the maximum value of y_i that is less than 1. The difference in the expected competitive ratios is then given by

$$\begin{aligned} & (1-\delta)\left(\frac{e}{e-1}\right)\left(\sum_{i=s}^{r-1}(1+y_s)\int_{y_i}^{y_{i+1}}\left(\frac{1}{y_i}-\frac{1}{x}\right)xe^{-x}dx + \int_{y_r}^{1}\left(\frac{1}{y_i}-\frac{1}{x}\right)xe^{-x}dx\right) \\ = \quad & \frac{e(1-\delta)}{e-1}\sum_{i=s}^{q}\int_{z_i}^{z_{i+1}}\left(\frac{x}{z_i}-1\right)e^{-x}dx, \end{aligned} \tag{1}$$

where $z_i = \min(y_i, 1)$.

We now have two cases. First, suppose $y_s > \frac{e}{2(e-1)}$. Then, the expected competitive ratio of strategy $S(y_s)$ when the input has distribution $\mathcal{D}$ is at least the expected competitive ratio when the input has distribution $\mathcal{D}_\delta$, which in turn is greater than $\frac{e}{e-1} + \frac{e-2}{2(e-1)}\delta$ by Lemma 3. Now, let $\delta = \frac{2(e-1)}{e-2}\varepsilon$; then the ratio is greater than $\frac{e}{e-1} + \varepsilon$.

The other case is when $y_s \leq \frac{e}{2(e-1)}$. Then, equation (1) and Lemma 3 imply that the expected competitive ratio of strategy $S(y_s)$ when the input has distribution $\mathcal{D}$ is at least

$$\begin{aligned} & \frac{e}{e-1} + \delta\left(y_s - \frac{1}{e-1}\right) + \frac{e(1-\delta)}{e-1}\sum_{i=s}^{q}\int_{z_i}^{z_{i+1}}\left(\frac{x}{z_i}-1\right)e^{-x}dx \\ \geq \quad & \frac{e}{e-1} - \frac{\delta}{e-1} + \frac{e(1-\delta)}{e-1}\sum_{i=s}^{q}\frac{e^{-z_i}}{z_i}\Delta_i^2, \quad \text{where } \Delta_i := z_{i+1} - z_i \geq 0 \\ \geq \quad & \frac{e}{e-1} - \frac{2\varepsilon}{e-2} + \frac{1}{2(e-1)}\sum_{i=s}^{q}\Delta_i^2 \\ \geq \quad & \frac{e}{e-1} - \frac{2\varepsilon}{e-2} + \frac{1}{2(e-1)}\frac{\left(\sum_{i=s}^{q}\Delta_i\right)^2}{q-s} \\ \geq \quad & \frac{e}{e-1} - \frac{2\varepsilon}{e-2} + \frac{(e-2)^2\varepsilon}{8(e-1)^3 C}. \end{aligned}$$

The first inequality follows from Fact 1; the second one from $z_i \leq 1$ and $\delta = \frac{2(e-1)}{e-2}\varepsilon$; the third one from $1 - \delta \geq 1/2$ since $\varepsilon \leq \frac{e-2}{4(e-1)}$; the fourth one from the fact that for any set of l numbers $d_1, \ldots, d_l$, $\sum_{i=1}^{l} d_i^2 \geq \frac{\left(\sum_{i=1}^{l} d_i\right)^2}{l}$; and the fifth one from $\sum_{i=s}^{q}\Delta_i \geq \frac{e-2}{2(e-1)}$ since $y_s \leq \frac{e}{2(e-1)}$, and from $q - s \leq q \leq C/\varepsilon$. Finally, we choose $C < \frac{1}{8e}\left(\frac{e-2}{e-1}\right)^3$ to ensure that the above competitive ratio is greater than $\frac{e}{e-1} + \varepsilon$. ◀

3 A Semi-stochastic Algorithm for the Ski-rental Problem

In this section, we will prove Theorem 2 by giving an algorithm that asks $O\left(\frac{1}{\epsilon^{3/2}}\right)$ quantile queries in the first stage, and then chooses a deterministic strategy (based on the quantiles revealed in the first stage) that has an expected competitive ratio of $\frac{e}{e-1} + \epsilon$. We describe the algorithm with a parameter δ that we will fix later. As mentioned in the introduction, we will assume wlog that the cost of buying skis is 1. Recall that any deterministic strategy can be described by a single parameter k which represents the strategy of renting skis until the duration of skiing x reaches k. At that point, skis are bought. As earlier, we denote this strategy by $S(k)$.

- In the first stage, the algorithm asks $1/\delta - 1$ queries for quantiles $\delta, 2\delta, \ldots, 1-\delta$. This partitions the input space $[0, \infty)$ into a set of $1/\delta$ contiguous, non-overlapping intervals, each of which has a probability mass of δ.
- The algorithm now assumes that the probability mass of each interval described above is concentrated at the right boundary of the interval, i.e. at its maximum value. For the last interval, this corresponds to assuming that there is δ mass of the input distribution at infinity.
- With the above assumption about the distribution, the algorithm outputs the deterministic strategy $S(k)$ that minimizes the expected competitive ratio subject to the constraint that $\delta^{1/3} \leq k \leq 1$. The only possible values of k are the values of the quantiles that are in the above range and $\delta^{1/3}$. Therefore, k can be found in time polynomial in $1/\delta$.

We will now analyze the expected competitive ratio of this strategy $S(k)$. We will use the following fact.

▶ Fact 2. For any $0 \leq a \leq 1$, $1 + \frac{1}{(1+a)e^{1-a}-1} \leq \frac{e}{e-1}(1+a^2)$.

Proof. We have

$$1+\frac{1}{(1+a)e^{1-a}-1} = \frac{e(1-(1+a)e^{-\alpha})}{((1+a)e^{1-a}-1)(e-1)}+\frac{e}{e-1} \leq \frac{e}{e-1}\left(2-(1+a)e^{-a}\right) \leq \frac{e}{e-1}(1+a^2).$$

The penultimate step follows from that fact that $((1+a)e^{1-a}-1) \geq 1$ for $0 \leq a \leq 1$, while the last step follows from the fact that for any z, $e^{-z} \geq 1 - z$. ◀

The next theorem is crucial.

▶ **Theorem 4.** *For* any *distribution over the input, the best deterministic strategy $S(k)$ subject to the constraint that $\alpha \leq k \leq 1$ for some fixed $0 \leq \alpha \leq 1$ has an expected competitive ratio of at most $\frac{e}{e-1}(1+\alpha^2)$.*

Proof. Let the worst-case input distribution for an algorithm that selects the best strategy in the above range (i.e. the input distribution for which the competitive ratio of the algorithm is the worst) be called the *nemesis* distribution. Our goal is to construct a nemesis distribution and show that the expected competitive ratio of the algorithm is at most $\frac{e}{e-1}(1+\alpha^2)$ for this distribution.

Let $\mathcal{P} : [0, \infty) \to [0, 1]$ be the probability density function corresponding to the nemesis distribution. Our first claim is that the nemesis distribution has no probability mass in the range $[0, \alpha]$, i.e. $\int_0^\alpha \mathcal{P}(x)dx = 0$. Suppose not; then, let $\int_0^\alpha \mathcal{P}(x)dx = \delta > 0$. Let the expected competitive ratio of the algorithm for this input be r. Since the strategy $S(\gamma)$ for any $\gamma \geq \alpha$ is optimal for the probability mass of δ in the range $[0, \alpha]$, it follows that

$$\delta + \int_\alpha^\gamma \mathcal{P}(x)dx + (1+\gamma)\int_\gamma^1 \frac{\mathcal{P}(x)}{x}dx + (1+\gamma)\int_1^\infty \mathcal{P}(x)dx \geq r$$

for all $\alpha \leq \gamma \leq 1$. Now, consider an alternative distribution which is identical to the previous distribution, except that this probability mass of δ is shifted from the range $[0, \alpha]$ to the range $[1, \infty]$ (the exact change in the density function can be arbitrary as long as the above property is satisfied). Now, for any strategy $S(\gamma)$ satisfying $\alpha \leq \gamma \leq 1$ that the algorithm can produce, the competitive ratio is

$$\begin{aligned} & \int_\alpha^\gamma \mathcal{P}(x)dx + (1+\gamma)\int_\gamma^1 \frac{\mathcal{P}(x)}{x}dx + (1+\gamma)\int_1^\infty \mathcal{P}(x)dx + (1+\gamma)\delta \\ > \quad & \int_\alpha^\gamma \mathcal{P}(x)dx + (1+\gamma)\int_\gamma^1 \frac{\mathcal{P}(x)}{x}dx + (1+\gamma)\int_1^\infty \mathcal{P}(x)dx + \delta \\ \geq \quad & r. \end{aligned}$$

This contradicts the definition of a nemesis distribution; hence $\int_0^\alpha \mathcal{P}(x)dx = 0$.

Let $\mathcal{P}_\infty = \int_1^\infty \mathcal{P}(x)dx$. Then the expected competitive ratio of strategy $S(\gamma)$ (where $\alpha \leq \gamma \leq 1$) can be thought of as a function of γ,

$$r(\gamma) = \int_\alpha^\gamma \mathcal{P}(x)dx + (1+\gamma)\int_\gamma^1 \frac{\mathcal{P}(x)}{x}dx + (1+\gamma)\mathcal{P}_\infty.$$

It follows from standard arguments (see e.g. [15, 14]) that for a nemesis distribution, $r(\gamma)$ must be invariant of γ. Let $r'(\gamma) = \frac{dr}{d\gamma}$ and $r''(\gamma) = \frac{d^2r}{d\gamma^2}$. Then,

$$\begin{aligned} r'(\gamma) &= \int_\gamma^1 \frac{\mathcal{P}(x)}{x}dx - \frac{\mathcal{P}(\gamma)}{\gamma} + \mathcal{P}_\infty, \quad \text{and} \\ r''(\gamma) &= -\frac{\mathcal{P}(\gamma)}{\gamma} - \frac{\mathcal{P}'(\gamma)}{\gamma} + \frac{\mathcal{P}(\gamma)}{\gamma^2}, \end{aligned}$$

where $\mathcal{P}'(\gamma) = \frac{d\mathcal{P}}{d\gamma}$. For the invariance property to hold, $r''(\gamma) = r'(\gamma) = 0$, which gives $\mathcal{P}(\gamma) = \mathcal{P}_\infty \gamma e^{1-\gamma}$. Then, the overall probability mass is given by

$$\int_0^\infty \mathcal{P}(x)dx = e\mathcal{P}_\infty \int_\alpha^1 xe^{-x}dx + \mathcal{P}_\infty = -e\mathcal{P}_\infty\left(\frac{2}{e} - (1+\alpha)e^{-\alpha}\right) + \mathcal{P}_\infty = \mathcal{P}_\infty\left((1+\alpha)e^{1-\alpha} - 1\right).$$

Since $\int_0^\infty \mathcal{P}(x)dx = 1$, it follows that $\mathcal{P}_\infty = \frac{1}{(1+\alpha)e^{1-\alpha}-1}$. Therefore, the probability density function for the nemesis distribution is

$$\mathcal{P}(x) = \begin{cases} 0 & \text{if } x < \alpha \\ \left(\frac{1}{(1+\alpha)e^{1-\alpha}-1}\right) xe^{1-x} & \text{if } \alpha \leq x \leq 1. \end{cases}$$

There is also a probability mass of $\frac{1}{(1+\alpha)e^{1-\alpha}-1}$ in the range $[1, \infty]$; the exact density function in this range is inconsequential.

Note that by our derivation, the expected competitive ratio of any strategy in the range $[\alpha, 1]$ is identical; hence, the expected competitive ratio of the strategy $S(k)$ chosen by the algorithm is given by (we calculate the competitive ratio of $S(1)$)

$$1 - \mathcal{P}_\infty + 2\mathcal{P}_\infty = 1 + \mathcal{P}_\infty = 1 + \frac{1}{(1+\alpha)e^{1-\alpha}-1} \leq \frac{e}{e-1}(1+\alpha^2),$$

by Fact 2. ◀

We now use this theorem to obtain a bound on the competitive ratio of the algorithm given above.

▶ **Theorem 5.** *The competitive ratio of the algorithm given above is at most* $\frac{e}{e-1} + \frac{2e-1}{e-1}\delta^{2/3}$.

Proof. Consider a game between the algorithm and an adversary where the adversary needs to initially present the quantiles queried by the algorithm, then the algorithm outputs a strategy $S(k)$ according to the description above, and finally the adversary reveals the actual distribution that is consistent with the quantiles it displayed initially. The goal of the adversary is to maximize the expected competitive ratio of the algorithm for the distribution that she reveals at the end. In the last step, the adversary has freedom to define any probability density function in the interval between every pair of adjacent quantiles as long as the total probability mass in any such interval is δ. We consider the options available to the adversary in the three ranges $[0, k]$, $[k, 1]$ and $[1, \infty)$. The actual probability density function in the first and last of these ranges does not change the expected competitive ratio

of the strategy $S(k)$. However, in the range $[k, 1]$, the adversary would move the entire probability mass in each individual interval to the quantile at its left boundary, thereby maximally increasing the expected competitive ratio of $S(k)$. This would decrease the cost of the optimal offline solution to the maximum extent while not changing the cost of the algorithmic solution. The net effect of these changes is that the actual input distribution has a δ probability mass at k instead of at 1. Thus, the difference of the expected competitive ratio of strategy $S(k)$ for the actual input distribution and that for the assumed distribution is at most $\delta\left(\frac{1+k}{k} - (1+k)\right) \leq \frac{\delta}{k} \leq \delta^{2/3}$ since $k \geq \delta^{1/3}$. By the above theorem, the expected competitive ratio of strategy $S(k)$ for the assumed distribution is at most $\frac{e}{e-1}(1+\delta^{2/3})$. Therefore, the competitive ratio of the algorithm for the actual input distribution is at most $\frac{e}{e-1} + \frac{2e-1}{e-1}\delta^{2/3}$. ◀

Finally, we note that Theorem 2 follows from the above theorem by replacing δ with $\left[\left(\frac{e-1}{2e-1}\right)\epsilon\right]^{3/2}$.

4 Conclusion and Future Work

In this paper, we introduced a new model for online optimization problems that we call the semi-stochastic model. This framework offers a hybrid between the excessively pessimistic online model and the overly optimistic stochastic model. As a first step towards understanding optimization problems in this setting, we presented an algorithm for the ski-rental problem that gives a trade-off between the amount of stochastic information available to the algorithm and its performance. Perhaps more surprisingly, we also gave a lower bound on the query complexity for algorithms in this model that loosely matches the upper bound. Our lower bound holds for the powerful class of adaptive algorithms.

This initial result opens up the possibility of studying a plethora of online problems in the semi-stochastic model. Our model does not impose any restriction on the type of queries that the algorithm is allowed to ask of the adversary. So, it would be interesting to consider other query models as well. For example, as discussed in the introduction, in certain (e.g. computational learning) situations, natural queries might correspond to sampling inputs.

A somewhat different direction of future research would be to investigate the robustness of stochastic algorithms. One interpretation of our semi-stochastic algorithms is that they are robust to changes in the input distribution modulo the satisfaction of a set of parameters (corresponding to the answers given to the queries). Following up in this direction, it would be interesting to investigate the sensitivity of known stochastic algorithms to changes in the input distribution, and if they turn out to be sensitive, to design algorithms that are robust to such changes. The ultimate goal in this direction of research would be to obtain algorithms that degrade gracefully with changes in the input distribution, with the best stochastic algorithm and the best online algorithm being the two ends of the spectrum.

5 Acknowledgement

We would like to thank Sudipto Guha, Adam Kalai, Kamesh Munagala and R. Ravi for useful discussions on our problem. We also thank an anonymous referee for detailed comments that helped improve the presentation of the paper.

References

1 Susanne Albers and Stefano Leonardi. On-line algorithms. *ACM Comput. Surv.*, 31(3es):4, 1999.
2 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *SODA*, pages 434–443, 2007.
3 Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When LP is the cure for your matching woes: Improved bounds for stochastic matchings - (extended abstract). In *ESA (2)*, pages 218–229, 2010.
4 Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.
5 Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *ICALP (1)*, pages 266–278, 2009.
6 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *SODA*, pages 395–404, 2005.
7 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008.
8 Jon Feldman, Aranyak Mehta, Vahab S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *FOCS*, pages 117–126, 2009.
9 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *SODA*, pages 942–951, 2008.
10 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, pages 982–991, 2008.
11 Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. In *FOCS*, pages 347–356, 2008.
12 Sudipto Guha and Kamesh Munagala. Approximation algorithms for budgeted learning problems. In *STOC*, pages 104–113, 2007.
13 Sudipto Guha and Kamesh Munagala. Model-driven optimization using adaptive probes. In *SODA*, pages 308–317, 2007.
14 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgment and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003.
15 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.
16 Robert Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM*, pages 1902–1909, 2007.
17 Robert D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, pages 630–631, 2005.
18 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *ICALP (2)*, pages 508–520, 2009.
19 Adam Meyerson. Online facility location. In *FOCS*, pages 426–431, 2001.
20 Adam Meyerson, Kamesh Munagala, and Serge A. Plotkin. Designing networks incrementally. In *FOCS*, pages 406–415, 2001.
21 Chaitanya Swamy and David B. Shmoys. Approximation algorithms for 2-stage stochastic optimization problems. *SIGACT News*, 37(1):33–46, 2006.
22 Wikipedia. Dirac delta function — Wikipedia, the free encyclopedia.
23 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227, 1977.

Streamability of Nested Word Transductions*

Emmanuel Filiot[1], Olivier Gauwin[2], Pierre-Alain Reynier[3], and Frédéric Servais[4]

1 **Université Libre de Bruxelles**
2 **Université de Mons**
3 **LIF, Aix-Marseille Univ & CNRS, France**
4 **Hasselt University and Transnational University of Limburg**

Abstract

We consider the problem of evaluating in streaming (*i.e.* in a single left-to-right pass) a nested word transduction with a limited amount of memory. A transduction T is said to be height bounded memory (HBM) if it can be evaluated with a memory that depends only on the size of T and on the height of the input word. We show that it is decidable in coNPTime for a nested word transduction defined by a visibly pushdown transducer (VPT), if it is HBM. In this case, the required amount of memory may depend exponentially on the height of the word. We exhibit a sufficient, decidable condition for a VPT to be evaluated with a memory that depends quadratically on the height of the word. This condition defines a class of transductions that strictly contains all determinizable VPTs.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases nested word, visibly pushdown transducer, streaming, XML

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.312

1 Introduction

Memory analysis is an important tool for ensuring system robustness. In this paper we focus on the analysis of programs processing *nested words* [2], *i.e.*, words with a recursive structure, like program traces, XML documents, or more generally unranked trees. On huge inputs, a *streaming* mode is often used, where the nested word is read only once, from left to right. This corresponds to a depth-first left-to-right traversal when the nested word is considered as a tree. For such programs, *dynamic analysis* problems have been addressed in various contexts. For instance, runtime verification detects dynamically, and as early as possible, whether a property is satisfied by a program trace [17, 6]. On XML streams, some algorithms outputting nodes selected by an XPath expression at the earliest possible event have also been proposed [7, 12]. These algorithms allow minimal buffering [3].

In this paper, we investigate *static analysis* of memory usage for a special kind of programs on nested words, namely programs defined by *transducers*. We assume that the transducers are *functional* and *non-deterministic*. Non-determinism is required as input words are read from left to right in a single pass and some actions may depend on the future of the stream. For instance, the XML transformation language XSLT uses XPath for selecting nodes where local transformations are applied, and XPath queries relies on

* Partially supported by the ESF project GASICS, by the FNRS, by the ANR project ECSPER (JC09-472677), by the PAI program Moves funded by the Federal Belgian Government and by the FET project FOX (FP7-ICT-233599).

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 312–324

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

non-deterministic moves along tree axes, such as a move to any descendant. We require our transducers to be *functional*, as we are mainly interested by transformation languages like XSLT, XQuery and XQuery Update Facility, for which any transformation maps each XML input document to a unique output document.

Visibly pushdown transducers (VPTs) form a subclass of pushdown transducers adequate for dealing with nested words and streaming evaluation, as the input nested word is processed from left to right. They are visibly pushdown automata [2] extended with arbitrary output words on transitions. VPTs capture interesting fragments of the aforementioned XML transformation languages that are amenable to efficient streaming evaluation, such as all editing operations (insertion, deletion, and relabeling of nodes, as used for instance in XQuery Update Facility) under all regular tests. Like for visibly pushdown automata, the stack behavior of VPTs is imposed by the type of symbols read by the transducer. Those restrictions on stack operations allow to decide functionality and equivalence of functional VPTs in PTime and ExpTime respectively [11].

Some transductions defined by (functional and non-deterministic) VPTs cannot be evaluated efficiently in streaming. For instance, swapping the first and last letter of a word can be defined by a VPT as follows: guess the last letter and transform the first letter into the guessed last letter, keep the value of the first letter in the state, and transform any value in the middle into itself. Any deterministic machine implementing this transformation requires to keep the entire word in memory until the last letter is read. It is not reasonable in practice as for instance XML documents can be very huge.

Our aim is thus to identify decidable classes of transductions for various memory requirements that are suitable to space-efficient streaming evaluation. We first consider the requirement that a transducer can be implemented by a program using a *bounded memory* (BM), *i.e.* computing the output word using a memory independent of the size of the input word. However when dealing with nested words in a streaming setting, the bounded memory requirement is quite restrictive. Indeed, even performing such a basic task as checking that a word is well-nested or checking that a nested word belongs to a regular language of nested words requires a memory dependent on the height (the level of nesting) of the input word [19]. This observation leads us to the second question: decide, given a transducer, whether the transduction can be evaluated with a memory that depends only on the size of the transducer and the height of the word (but not on its length). In that case, we say that the transduction is *height bounded memory* (HBM). This is particularly relevant to XML transformations as XML documents can be very long but have usually a small depth [5]. HBM does not specify *how* memory depends on the height. A stronger requirement is thus to consider HBM transductions whose evaluation can be done with a memory that depends *polynomially* on the height of the input word.

Contributions First, we give a general space-efficient evaluation algorithm for functional VPTs. After reading a prefix of an input word, the number of configurations of the (non-deterministic) transducer as well as the number of output candidates to be kept in memory may be exponential in the size of the transducer and the height of the input word (but not in its length). Our algorithm produces as output the longest common prefix of all output candidates, and relies on a compact representation of sets of configurations and remaining output candidates (the original output word without the longest common prefix). We prove that it uses a memory linear in the height of the input word, and linear in the maximal length of a remaining output candidate.

We prove that BM is equivalent to subsequentiability for finite state transducers (FSTs), which is known to be decidable in PTime. BM is however undecidable for arbitrary push-

down transducers but we show that it is decidable for VPTs in coNPTime.

Like BM, HBM is undecidable for arbitrary pushdown transductions. We show, via a non-trivial reduction to the emptiness of pushdown automata with bounded reversal counters, that it is decidable in coNPTime for transductions defined by VPTs. In particular, we show that the previously defined algorithm runs in HBM iff the VPT satisfies some property, which is an extension of the so called *twinning property* for FSTs [9] to nested words. We call it the *horizontal twinning property*, as it only cares about configurations of the transducers with stack contents of identical height. This property only depends on the transduction, *i.e.* is preserved by equivalent transducers.

When a VPT-transduction is height bounded memory, the memory needed may be exponential in the height of the word. We introduce a refinement of the twinning property that takes the height of the configurations into account, hence called *matched twinning property*. A VPT satisfying this property is called *twinned*. We prove that the evaluation of *twinned transductions* with our algorithm uses a memory *quadratic* in the height of the input word. We show that it is decidable in coNPTime whether a VPT is twinned. Moreover, the most challenging result of this paper is to show that being twinned depends only on the transduction and not on the VPT that defines it. Thus, this property indeed defines a class of transductions. As a consequence of this result, all subsequentializable VPTs are twinned, because subsequential VPTs trivially satisfy the matched twinning property. The class of twinned transductions captures a strictly larger class than subsequentializable VPTs while staying in the same complexity class for evaluation, i.e. polynomial space in the height of the input word when the transducer is fixed.

Related Work In the XML context, visibly pushdown automata based streaming processing has been extensively studied for validating XML streams [16, 4, 19]. The validation problem with bounded memory is studied in [4] when the input is assumed to be a well-nested word and in [19] when it is assumed to be a well-formed XML document (this problem is still open). Querying XML streams has been considered in [13]. It consists in selecting a set of tuples of nodes in the tree representation of the XML document. For monadic queries (selecting nodes instead of tuples), this can be achieved by a functional VPT returning the input stream of tags, annotated with Booleans indicating selection by the query. However, functional VPTs cannot encode queries of arbitrary arities. The setting for functional VPTs is in fact different to query evaluation, because the output has to be produced on-the-fly in the right order, while query evaluation algorithms can output nodes in any order: an incoming input symbol can be immediately output, while another candidate is still to be confirmed. This makes a difference with the notion of concurrency of queries, measuring the minimal amount of candidates to be stored, and for which algorithms and lower bounds have been proposed [3]. VPTs also relate to tree transducers [11], for which no comparable work on memory requirements is known. However, the height of the input word is known to be a lower bound for Core XPath filters [13]. As VPTs can express them, this lower bound also applies when evaluating VPTs. When allowing two-way access on the input stream, space-efficient algorithms for XML validation [15] and querying [18] have been proposed.

2 Visibly Pushdown Languages and Transductions

Words and nested words In this paper, we consider nested words accessed in streaming. Their nesting structure is thus discovered on-the-fly, so we consider a finite alphabet Σ partitioned into three disjoint sets Σ_c, Σ_r and Σ_ι, denoting respectively the *call*, *return* and *internal* alphabets. We denote by Σ^* the set of (finite) words over Σ and by ϵ the empty

word. The length of a word u is denoted by $|u|$. For all words $u, v \in \Sigma^*$, we denote by $u \wedge v$ the longest common prefix of u and v. More generally, for any non-empty finite set of words $V \subseteq \Sigma^*$, the longest common prefix of V, denoted by $\mathsf{lcp}(V)$, is inductively defined by $\mathsf{lcp}(\{u\}) = u$ and $\mathsf{lcp}(V \cup \{u\}) = \mathsf{lcp}(V) \wedge u$. The set of *well-nested* words Σ^*_{wn} is the smallest subset of Σ^* such that $\Sigma^*_\iota \subseteq \Sigma^*_{\mathsf{wn}}$ and for all $c \in \Sigma_c$, all $r \in \Sigma_r$, all $u, v \in \Sigma^*_{\mathsf{wn}}$, $cur \in \Sigma^*_{\mathsf{wn}}$ and $uv \in \Sigma^*_{\mathsf{wn}}$. Let $u = \alpha_1 \dots \alpha_n \in \Sigma^*$ be a prefix of a well-nested word. A position $i \in \{1, \dots, n\}$ is a *pending call* if $\alpha_i \in \Sigma_c$ and for all $j \geq i$, $\alpha_i \dots \alpha_j \notin \Sigma^*_{\mathsf{wn}}$. The *height* of u is the maximal number of pending calls on any prefix of u, i.e.

$$h(u) = \max_{1 \leq i \leq n} |\{k \mid 1 \leq k \leq i, \alpha_k \text{ is a pending call of } \alpha_1 \dots \alpha_i\}|$$

For instance, $h(crcrcc) = h(ccrcrr) = 2$. In particular, for well-nested words, the height corresponds to the usual height of the nesting structure of the word.

Given two words $u, v \in \Sigma^*$, the *delay* of u and v, denoted by $\Delta(u, v)$, is the unique pair of words (u', v') such that $u = (u \wedge v)u'$ and $v = (u \wedge v)v'$. For instance, $\Delta(abc, abde) = (c, de)$. Informally, in a word transduction, if there are two output candidates u and v during the evaluation, we are sure that we can output $u \wedge v$ and $\Delta(u, v)$ is the remaining suffixes we still keep in memory.

Visibly pushdown transducers (VPTs) As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend visibly pushdown automata [2] with outputs [11]. To simplify notations, we suppose that the output alphabet is Σ, but our results still hold for an arbitrary output alphabet. Informally, the stack behavior of a VPT is similar to the stack behavior of visibly pushdown automata (VPA). On a call symbol, the VPT pushes a symbol on the stack and produces some output word (possibly empty), on a return symbol, it must pop the top symbol of the stack and produce some output word (possibly empty) and on an internal symbol, the stack remains unchanged and it produces some output word. Formally:

▶ **Definition 1.** A *visibly pushdown transducer* (VPT) on finite words over Σ is a tuple $T = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ the set of final states, Γ is the stack alphabet, $\delta = \delta_c \uplus \delta_r \uplus \delta_\iota$ the (finite) transition relation, with $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times \Gamma \times Q$, $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$, and $\delta_\iota \subseteq Q \times \Sigma_\iota \times \Sigma^* \times Q$.

A *configuration* of a VPT is a pair $(q, \sigma) \in Q \times \Gamma^*$. A *run* of T on a word $u = a_1 \dots a_l \in \Sigma^*$ from a configuration (q, σ) to a configuration (q', σ') is a finite sequence $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$ such that $q_0 = q$, $\sigma_0 = \sigma$, $q_l = q'$, $\sigma_l = \sigma'$ and for each $1 \leq k \leq l$, there exist $v_k \in \Sigma^*$ and $\gamma_k \in \Gamma$ such that either $(q_{k-1}, a_k, v_k, \gamma_k, q_k) \in \delta_c$ and $\sigma_k = \sigma_{k-1}\gamma_k$ or $(q_{k-1}, a_k, v_k, \gamma_k, q_k) \in \delta_r$ and $\sigma_{k-1} = \sigma_k\gamma_k$, or $(q_{k-1}, a_k, v_k, q_k) \in \delta_\iota$ and $\sigma_k = \sigma_{k-1}$. The word $v = v_1 \dots v_l$ is called an *output* of ρ. We write $(q, \sigma) \xrightarrow{u/v} (q', \sigma')$ when there exists a run on u from (q, σ) to (q', σ') producing v as output. We denote by $\perp$ the empty word on Γ. A configuration (q, σ) is *accessible* (resp. is *co-accessible*) if there exist $u, v \in \Sigma^*$ and $q_0 \in I$ (resp. $q_f \in F$) such that $(q_0, \perp) \xrightarrow{u/v} (q, \sigma)$ (resp. such that $(q, \sigma) \xrightarrow{u/v} (q_f, \perp)$). A transducer T is *reduced* if every accessible configuration is co-accessible. Given any VPT, computing an equivalent reduced VPT can be performed in polynomial time [8][1]. A transducer T defines the binary word relation $[\![T]\!] = \{(u, v) \in \Sigma^* \times \Sigma^* \mid \exists q \in I, q' \in F,\ (q, \perp) \xrightarrow{u/v} (q', \perp)\}$.

A *transduction* is a binary relation $R \subseteq \Sigma^* \times \Sigma^*$. We say that a transduction R is a VPT-transduction if there exists a VPT T such that $R = [\![T]\!]$. For any input word $u \in \Sigma^*$, we denote by $R(u)$ the set $\{v \mid (u, v) \in R\}$. Similarly, for a VPT T, we denote by $T(u)$ the

[1] The reduction of VPAs in [8] trivially extends to VPTs.

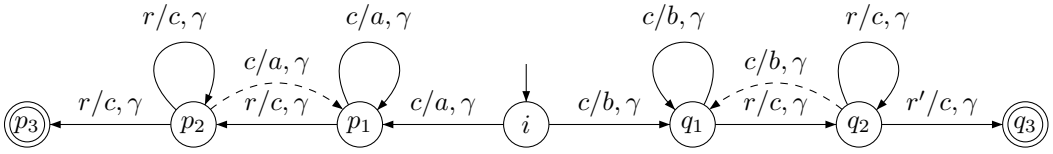


Figure 1 A functional VPT with $\Sigma_c = \{c\}$, $\Sigma_r = \{r, r'\}$ and $\Sigma_\iota = \{a, b\}$

set $[\![T]\!](u)$. A transduction R is *functional* if for all $u \in \Sigma^*$, $R(u)$ has size at most one. If R is functional, we identify $R(u)$ with the unique image of u if it exists. A VPT T is functional if $[\![T]\!]$ is functional, and this can be decided in PTime [11]. The class of functional VPTs is denoted by fVPT. The *domain* of T (denoted by $Dom(T)$) is the domain of $[\![T]\!]$. The domain of T contains only well-nested words, which is not necessarily the case of the codomain.

▶ **Example 2.** Consider the VPT T of Fig. 1 represented in plain arrows. The left and right parts accept the same input words except for the last letter of the word. The domain of T is $Dom(T) = \{c^n r^n \mid n \geq 2\} \cup \{cc^n r^n r' \mid n \geq 1\}$. Any word $c^n r^n$ is translated into $a^n c^n$, and any word $cc^n r^n r'$ is translated into $b^{n+1} c^{n+1}$. Therefore the translation of the first sequence of calls depends on the last letter r or r'. This transformation cannot be evaluated with a bounded amount of memory, but with a memory which depends on the height n of the input word.

Finite state transducers (FSTs) A *finite state transducer* (FST) on an alphabet Σ is a tuple (Q, I, F, δ) where Q is a finite set, $I, F \subseteq Q$ and $\delta \subseteq Q \times \Sigma \times \Sigma^* \times Q$ with the standard semantics. This definition corresponds to the usual definition of *real-time* FSTs, as there is no ϵ-transitions. We always consider real-time FSTs in this paper, so we just call them FSTs.

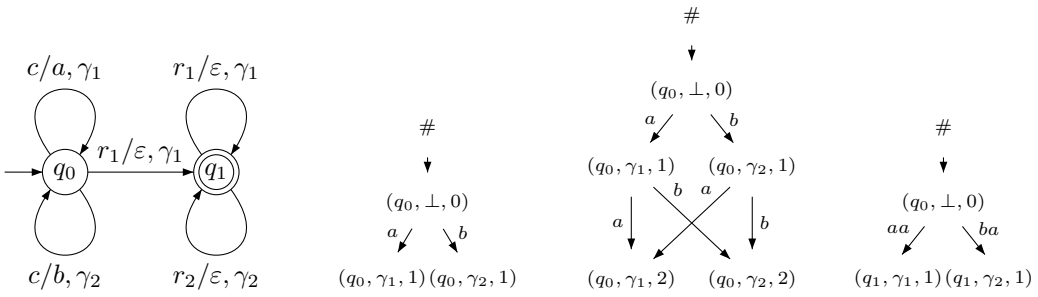
A *subsequential* FST (resp. VPT) is a pair (T, Ψ) where T is an (input) deterministic FST (resp. VPT) and $\Psi : F \to \Sigma^*$. The outputs of u by (T, Ψ) are the words $v.\Psi(q)$ whenever there is a run of T on u producing v and ending up in some accepting state q.

Given an integer $k \in \mathbb{N}$ and a VPT T, one can define an FST, denoted by $\mathsf{FST}(T, k)$, which is the restriction of T to input words of height less than k. The transducer is naturally constructed by taking as states the configurations (q, σ) of T such that $|\sigma| \leq k$.

Turing Transducers In order to formally define the complexity classes for evaluation that we target, we introduce a *deterministic* computational model for word transductions that we call *Turing Transducers*. Turing transducers have three tapes: one read-only left-to-right input tape, one write-only left-to-right output tape, and one standard working tape. Such a machine naturally defines a transduction: the input word is initially on the input tape, and the result of the transduction is the word written on the output tape after the machine terminates in an accepting state. We denote by $[\![M]\!]$ the transduction defined by M. The space complexity is measured on the working tape only.

3 Online Evaluation Algorithm of VPT-Transductions

We present an online algorithm LcpIn to evaluate functional word transductions defined by fVPTs. For clarity, we present this algorithm under some assumptions, without loss of generality. First, input words of our algorithms are words $u \in \Sigma^*$ concatenated with a special symbol $\$ \notin \Sigma$, denoting the end of the word. Second, we only consider input words without internal symbols, as they can easily be encoded by successive call and return symbols. Third, input words are supposed to be valid, in the sense that they produce an output. It is indeed easy to extend our algorithms in order to raise an error message when the input is not in the domain, *i.e.* when no run of the VPT applies on the input.



(a) VPT T_1. **(b)** After reading c. **(c)** After reading cc. **(d)** After reading ccr_1.

Figure 2 Data structure used by LcpIn.

The core task of this algorithm is to maintain the configuration for each run of the fVPT T on the input u, and produce its output on-the-fly. Therefore, the algorithm LcpIn only applies on reduced fVPTs. Indeed, as T is reduced, functionality ensures that, for a given input word u, and for every accessible configuration (q, σ) of T, there is at most one v such that $(q_i, \perp) \xrightarrow{u/v} (q, \sigma)$ with $q_i \in I$. Hence, a configuration is a triple (q, σ, w) where q is the current state of the run, σ its corresponding stack content, and w the part of the output that has been read but not output yet. We call such a configuration *d-configuration* and write $\mathsf{Dconfs}(T) = Q \times \Gamma^* \times \Sigma^*$ for the set of d-configurations of T. Algorithm LcpIn relies on two main features.

Compact representation First, the set of current d-configurations is stored in a compact structure that shares common stack contents. Consider for instance the VPT T_1 in Fig. 2a. After reading cc, current d-configurations are $\{(q_0, \gamma_1\gamma_1, aa), (q_0, \gamma_1\gamma_2, ab), (q_0, \gamma_2\gamma_1, ba), (q_0, \gamma_2\gamma_2, bb)\}$. Hence after reading c^n, the number of current d-configurations is 2^n. However, the transition used to update a d-configuration relates the stack symbol and the output word. For instance, the previous set is the set of tuples $(q_0, \eta_1\eta_2, \alpha_1\alpha_2)$ where (η_i, α_i) is either (γ_1, a) or (γ_2, b). Based on this observation, we propose a data structure avoiding this blowup. As illustrated in Fig. 2b to 2d, this structure is a directed acyclic graph (DAG). Nodes of this DAG are tuples (q, γ, i) where $q \in Q$, $\gamma \in \Gamma$ and $i \in \mathbb{N}$ is the depth of the node in the DAG. Each edge of the DAG is labelled with a word, so that a branch of this DAG, read from the root # to the leaf, represents a d-configuration (q, σ, v): q is the state in the leaf, σ is the concatenation of stack symbols in traversed nodes, and v is the concatenation of words on edges. For instance, in the DAG of Fig. 2c, the branch $\# \to (q_0, \perp, 0) \xrightarrow{b} (q_0, \gamma_2, 1) \xrightarrow{a} (q_0, \gamma_1, 2)$ encodes the d-configuration $(q_0, \gamma_2\gamma_1, ba)$ of the VPT of Fig. 2.(a). However, this data structure cannot store any set of accessible d-configurations of arbitrary functional VPTs: at most one delay w has to be assigned to a d-configuration. This is why we need T to be reduced.

Computing outputs Second, after reading a prefix u' of a word u, LcpIn will have output the common prefix of all corresponding runs, i.e. $\mathsf{lcp_{in}}(u', T) = \mathsf{lcp}(\mathsf{reach}(u'))$ where $\mathsf{reach}(u') = \{v \mid \exists (q_0, q, \sigma) \in I \times Q \times \Gamma^*,\ (q_0, \perp) \xrightarrow{u'/v} (q, \sigma)\}$. When a new input symbol is read, the DAG is first updated. Then, a bottom-up pass on this DAG computes $\mathsf{lcp_{in}}(u', T)$ in the following way. For each node, let ℓ be the largest common prefix of labels of outgoing edges. Then ℓ is removed from these outgoing edges, and concatenated at the end of labels of incoming edges. At the end, the largest common prefix of all output words on branches is the largest common prefix of words on edges outgoing from the root node #.

Let $\mathsf{out}_{\neq}(u')$ be the maximal size of outputs of T on u' where their common prefix is

removed: $\mathsf{out}_{\neq}(u') = \max_{v \in \mathsf{reach}(u')} |v| - |\mathsf{lcp}_{\mathsf{in}}(u', T)|$ and $\mathsf{out}^{\max}_{\neq}(u)$ its maximal value over prefixes of u: $\mathsf{out}^{\max}_{\neq}(u) = \max_{u' \text{ prefix of } u} \mathsf{out}_{\neq}(u')$. To summarize, one can in polynomial time reduce T if necessary, and then build the Turing transducer associated with the algorithm LcpIn. We prove the following complexity result:

▶ **Proposition 3.** *Given an* fVPT T*, one can build in* PTime *a Turing transducer, denoted* LcpInTT(T)*, which, on an input stream* $u \in \Sigma^*$*, runs in space complexity* $O((h(u)+1) \cdot \mathsf{out}^{max}_{\neq}(u))$.

In addition, when T is reduced, we can detail how the constant depends on the size of T. The space used by LcpInTT(T) for computing $T(u)$ is in $O(|Q|^2 \cdot |\Gamma|^2 \cdot (h(u)+1) \cdot \mathsf{out}^{\max}_{\neq}(u))$.

4 Bounded Memory Evaluation Problems

Bounded Memory Transductions

We first consider transductions that can be evaluated with a constant amount of memory if we fix the machine that defines the transduction:

▶ **Definition 4.** A (functional) transduction $R \subseteq \Sigma^* \times \Sigma^*$ is *bounded memory (BM)* if there exists a Turing transducer M and $K \in \mathbb{N}$ such that $[\![M]\!] = R$ and on any input word $u \in \Sigma^*$, M runs in space complexity at most K.

It is not difficult (see [10]) to verify that for FST-transductions, bounded memory is characterized by subsequentializability, which is decidable in PTime [20]. Moreover, BM is undecidable for pushdown transducers, since it is as difficult as deciding whether a pushdown automaton defines a regular language. For VPTs, BM is quite restrictive as it imposes to verify whether a word is well-nested by using a bounded amount of memory. This can be done only if the height of the words of the domain is bounded by some constant which depends on the transducer only:

▶ **Proposition 5.** *Let* T *be a functional* VPT *with* n *states.*

1. $[\![T]\!]$ *is BM iff* (i) *for all* $u \in Dom(T)$, $h(u) \leq n^2$, *and* (ii) $\mathsf{FST}(T, n^2)$ *is BM;*
2. *It is decidable in* coNPTime *whether* $[\![T]\!]$ *is BM.*

Sketch. The first assertion is obvious by using simple pumping techniques to show that bounded memory implies bounded height. In the sequel, we define the class of height bounded memory transductions, and show it is decidable in coNPTime. On words of bounded height, this class collapses with bounded memory transductions. ◀

Height Bounded Memory Transductions

As we have seen, bounded memory is too restrictive to still benefit from the extra expressiveness of VPT compared to FST, namely the ability to recognize nested words of unbounded height. In this section, we define a notion of bounded memory which is well-suited to VPTs.

▶ **Definition 6.** A (functional) transduction $R \subseteq \Sigma^* \times \Sigma^*$ is *height bounded memory (HBM)* if there exists a Turing transducer M and a function $f : \mathbb{N} \to \mathbb{N}$ such that $[\![M]\!] = R$ and on any input word $u \in \Sigma^*$, M runs in space at most $f(h(u))$.

Note that this definition ensures that the machine cannot store all the input words on the working tape in general. The VPT in Fig. 2a is not in BM, but is in HBM: the stack content suffices (and is necessary) to determine the output. When the structured alphabet contains only internal letters, HBM and BM coincide, thus it is undecidable whether a pushdown transducer is HBM. The remainder of this section is devoted to the proof that HBM is decidable for fVPTs.

BM functional FST-transductions (or equivalently subsequentializable FSTs) are characterized by the so called *twinning property* [9], which is decidable in PTIME [20]. We introduce a similar characterization of HBM fVPTs-transductions, called the *horizontal twinning property* (HTP). The restriction of the horizontal twinning property to FSTs is equivalent to the usual twinning property for FSTs (see [10]). Intuitively, the HTP requires that two runs on the same input cannot accumulate increasing output delay on loops.

▶ **Definition 7.** Let T be an fVPT. T satisfies the *horizontal twinning property* (HTP) if for all $u_1, u_2, v_1, v_2, w_1, w_2 \in \Sigma^*$ such that u_2 is well-nested, for all $q_0, q_0' \in I$, for all $q, q' \in Q$, and for all $\sigma, \sigma' \in \Gamma^*$ such that (q, σ) and (q', σ') are co-accessible,

$$\text{if } \begin{cases} (q_0, \bot) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \\ (q_0', \bot) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \end{cases} (1) \quad \text{then } \Delta(v_1, w_1) = \Delta(v_1 v_2, w_1 w_2).$$

▶ **Example 8.** Consider the VPT of Fig. 1 (including dashed arrows). It does not satisfy the HTP, as the delays increase when looping on *crcr*... Without the dashed transitions, the HTP is satisfied.

▶ **Lemma 9.** *The HTP is decidable in* coNPTIME *for* fVPTs.

Proof. First, let us show that an fVPT T does not satisfy the HTP if and only if there exist $u_1, u_2, v_1, v_2, w_1, w_2 \in \Sigma^*$, $q_0, q_0' \in I$, $q, q' \in Q$, and $\sigma, \sigma' \in \Gamma^*$ such that (q, σ) and (q', σ') are co-accessible, satisfy (1), and such that either we have (i) $|v_2| \neq |w_2|$, or (ii) $|v_2| = |w_2|$, $|v_1| \leq |w_1|$ and not $v_1 v_2 \preceq w_1 w_2$. Indeed, one can easily check that it is a necessary condition. To prove that it is a sufficient condition, suppose we have elements that satisfy (1) with $\Delta(v_1, w_1) \neq \Delta(v_1 v_2, w_1 w_2)$ but conditions (i) and (ii) do not hold. Wlog, we can assume that $|v_1| \leq |w_1|$, therefore we have $|v_2| = |w_2|$, $|v_1| \leq |w_1|$, $v_1 v_2 \preceq w_1 w_2$ and $\Delta(v_1, w_1) \neq \Delta(v_1 v_2, w_1 w_2)$. One can verify (see [10]) that there exists $k \in \mathbb{N}$ such that replacing u_2 with $u_2' = {u_2}^k$ yields a system that satisfies (ii).

Second, let T be an fVPT, we define a pushdown automaton with bounded reversal counters [14], A, such that the language of A is empty if and only if T satisfies the HTP. More precisely, A accepts the words $u = u_1 u_2 u_3 \in Dom(T)$ such that there exist $v_1, v_2, w_1, w_2 \in \Sigma^*$, $q_0, q_0' \in I$, $q, q' \in Q$, and $\sigma, \sigma' \in \Gamma^*$ that satisfy (1) and either (i) or (ii). A simulates in parallel any two runs of T on the input word (product automaton). It guesses the end of u_1 and stores the states q and q' of the first and second run (in order to be able to check that the simulated runs of T are in state q, resp. q' after reading u_2). Non-deterministically, it checks whether (i) or (ii) holds. To check (i), it uses two counters, one for each run. It does so by, after reading u_1, increasing the counters by the length of the output word of each transition of the corresponding run. Then, when reaching the end of u_2 it checks that both counters are different (by decreasing in parallel both counters and checking they do not reach 0). Similarly, using two other counters, A checks that (ii) holds as follows. Note that condition (ii) implies that there is a position p such that the p-th letter a_1 of $v_1 v_2$ and the p-th letter a_2 of $w_1 w_2$ are different. The automaton A guesses the position $p \in \mathbb{N}$ of the mismatch, and initializes both counters to the value p. Then, while reading $u_1 u_2$, it decreases each counter by the length of the output words of the corresponding run. When

a counter reaches 0, A stores the output letter of the corresponding run. Finally, A checks that $a_1 \neq a_2$, and that both configurations are co-accessible. T satisfies the HTP iff the language of A is empty. The latter is decidable in coNPTime [11]. ◀

We now show that HTP characterizes HBM fVPTs-transductions and therefore by Lemma 9 we get:

▶ **Theorem 10.** *Let T be an* fVPT*. Then $[\![T]\!]$ is HBM iff the HTP holds for T, which is decidable in* coNPTime*. In this case, the Turing transducer* LcpInTT(T) *runs, on an input stream u, in space complexity exponential in the height of u.*

We can state more precisely the space complexity of LcpInTT(T) when T is reduced. In this case, it is in $O(|Q|^4 \cdot |\Gamma|^{2h(u)+2} \cdot (h(u)+1) \cdot M)$, where $M = \max\{|v| \mid (q, a, v, \gamma, q') \in \delta\}$.

Sketch. We prove that $[\![T]\!]$ is HBM iff the HTP holds for T. To prove that the HTP is a necessary condition to be in HBM, we proceed by contradiction. We find a counter-example for the HTP and we let K be the height of the input word of this counter-example. It implies that the twinning property for FSTs does not hold for $\mathsf{FST}(T, K)$, and therefore $\mathsf{FST}(T, K)$ is not BM by Proposition 5. In particular, T is not HBM.
For the converse, it can easily be shown that when T satisfies the HTP, the procedure of [8] that reduces T preserves the HTP satisfiability. In particular, there is a one-to-one mapping between the runs of T and the runs of its reduction that preserves the output words. We then show that for any input word $u \in \Sigma^*$, the maximal delay $\mathsf{out}_{\neq}^{\max}(u)$ between the outputs of u is bounded by $(|Q| \cdot |\Gamma|^{h(u)})^2 M$. This is done by a pumping technique "by width" that relies on the property $\Delta(vv', ww') = \Delta(\Delta(v, w) \cdot (v', w'))$ for any words v, v', w, w'. Thus for an input word for which there are two runs that pass by the same configurations twice at the same respective positions, the delay of the output is equal to the delay when removing the part in between the identical configurations. Finally we apply Proposition 3. ◀

HBM is tight Theorem 10 shows that the space complexity of a VPT in HBM is at most exponential. We give here an example illustrating the tightness of this bound. The idea is to encode the tree transduction $f(t, a) \mapsto f(t, a) \cup f(t, b) \mapsto f(\bar{t}, b)$ by a VPT, where t is a binary tree over $\{0, 1\}$ and $\bar{t}$ is the mirror of t, obtained by replacing the 0 by 1 and the 1 by 0 in t. Thus taking the identity or the mirror depends on the second child of the root f. To evaluate this transformation in a streaming manner, one has to store the whole subtree t in memory before deciding to transform it into t or $\bar{t}$. The evaluation of this transduction cannot be done in polynomial space as there are a doubly exponential number of trees of height n, for all $n \geq 0$.

HBM vs Subsequentializable fVPTs We have seen that a functional transduction defined by an FST T is BM iff T is subsequentializable. We give an example illustrating that for VPTs, being subsequentializable is too strong to characterize HBM. Consider the VPT of Fig. 1 defined by the plain arrows. The transduction it defines is in HBM by Proposition 3, as at any time the delay between two outputs is bounded by the height of the input: $\mathsf{out}_{\neq}^{\max}(u) \leq 2h(u)$. However it is not subsequentializable, as the transformation of c into a or b depends on the last return.

5 Quadratic Height Bounded Memory Evaluation

In the previous section, we have shown that a VPT-transduction is in HBM iff the horizontal twinning property holds, and if it is in HBM, the algorithm of Section 3 uses a memory

at most exponential in the height of the word, and this bound is tight. To avoid this exponential cost, we identify in this section a subclass of HBM containing transductions for which the evaluation algorithm of Section 3 uses a memory *quadratic in the height of the word*. Therefore, we strengthen the horizontal twinning property by adding some properties for well-matched loops. Some of our main and challenging results are to show the decidability of this property and that it depends only on the transduction, i.e. is preserved by equivalent transducers. We show that subsequential VPTs satisfy this condition and therefore our class *subsumes* the class of subsequentializable transducers.

The property is a strengthening of the horizontal twinning property that we call the *matched twinning property (MTP)*. Intuitively, the MTP requires that two runs on the same input cannot accumulate increasing output delay on well-matched loops. They can accumulate delay on loops with increasing stack but this delay has to be caught up on the matching loops with descending stack.

▶ **Definition 11.** Let $T = (Q, I, F, \Gamma, \delta)$ be an fVPT. T satisfies the *matched twinning property* (MTP) if for all $u_i, v_i, w_i \in \Sigma^*$ ($i \in \{1, \dots, 4\}$) such that u_3 is well-nested, and u_2u_4 is well-nested, for all $i, i' \in I$, for all $p, q, p', q' \in Q$, and for all $\sigma_1, \sigma_2 \in \bot.\Gamma^*$, for all $\sigma'_1, \sigma'_2 \in \Gamma^*$, such that (q, σ_1) and (q', σ_2) are co-accessible:

$$\text{if } \begin{cases} (i,\bot) \xrightarrow{u_1/v_1} (p,\sigma_1) \xrightarrow{u_2/v_2} (p,\sigma_1\sigma'_1) \xrightarrow{u_3/v_3} (q,\sigma_1\sigma'_1) \xrightarrow{u_4/v_4} (q,\sigma_1) \\ (i',\bot) \xrightarrow{u_1/w_1} (p',\sigma_2) \xrightarrow{u_2/w_2} (p',\sigma_2\sigma'_2) \xrightarrow{u_3/w_3} (q',\sigma_2\sigma'_2) \xrightarrow{u_4/w_4} (q',\sigma_2) \end{cases}$$

then $\Delta(v_1v_3, w_1w_3) = \Delta(v_1v_2v_3v_4, w_1w_2w_3w_4)$. We say that a VPT T is *twinned* whenever it satisfies the MTP.

Note that any twinned VPT also satisfies the HTP (with $u_3 = u_4 = \epsilon$).

▶ **Example 12.** The VPT of Fig. 1 with plain arrows does not satisfy the MTP, as the delay between the two branches increases when iterating the loops. Consider now the VPT obtained by replacing r by r' in the transition (q_1, r, c, γ, q_2). It is obviously twinned, as we cannot construct two runs on the same input which have the form given in the premises of the MTP. However this transducer is not subsequentializable, as the output on the call symbols cannot be delayed to the matching return symbols.

As for the HTP, we can decide the MTP using a reduction to the emptiness of a pushdown automaton with bounded reversal counters. A complete proof can be found in [10].

▶ **Lemma 13.** *The matched twinning property is decidable in* coNPTime *for* fVPTs.

The most challenging result of this paper is to show that the MTP only depends on the transduction and not on the transducer that defines it. The proof relies on fundamental properties of word combinatorics that allow us to give a general form of the output words $v_1, v_2, v_3, v_4, w_1, w_2, w_3, w_4$ involved in the MTP, that relates them by means of conjugacy of their primitive roots. The proof gives a deep insight into the expressive power of VPTs which is also interesting on its own. As many results of word combinatorics, the proof is a long case study, so that we give it in [10] only.

▶ **Theorem 14.** *Let T_1, T_2 be* fVPTs *such that $[\![T_1]\!] = [\![T_2]\!]$. T_1 is twinned iff T_2 is twinned.*

Sketch. We assume that T_1 is not twinned and show that T_2 is not twinned either. By definition of the MTP there are two runs of the form

$$\begin{cases} (i_1,\bot) \xrightarrow{u_1/v_1} (p_1,\sigma_1) \xrightarrow{u_2/v_2} (p_1,\sigma_1\beta_1) \xrightarrow{u_3/v_3} (q_1,\sigma_1\beta_1) \xrightarrow{u_4/v_4} (q_1,\sigma_1) \\ (i'_1,\bot) \xrightarrow{u_1/v'_1} (p'_1,\sigma'_1) \xrightarrow{u_2/v'_2} (p'_1,\sigma'_1\beta'_1) \xrightarrow{u_3/v'_3} (q'_1,\sigma'_1\beta'_1) \xrightarrow{u_4/v'_4} (q'_1,\sigma'_1) \end{cases}$$

such that (q_1, σ_1) and (q'_1, σ'_1) are co-accessible and $\Delta(v_1v_3, v'_1v'_3) \neq \Delta(v_1v_2v_3v_4, v'_1v'_2v'_3v'_4)$. We will prove that by pumping the loops on u_2 and u_4 sufficiently many times we will get a similar situation in T_2, proving that T_2 is not twinned. It is easy to show that there exist $k_2 > 0$, $k_1, k_3 \geq 0$, $w_i, w'_i \in \Sigma^*$, $i \in \{1, \dots, 4\}$, some states $i_2, p_2, q_2, i'_2, p'_2, q'_2$ of T_2 and some stack contents $\sigma_2, \beta_2, \sigma'_2, \gamma'_2$ of T_2 such that we have the following runs in T_2:

$$\begin{cases} (i_2, \bot) \xrightarrow{u_1u_2^{k_1}/w_1} (p_2, \sigma_2) \xrightarrow{u_2^{k_2}/w_2} (p_2, \sigma_2\beta_2) \xrightarrow{u_2^{k_3}u_3u_4^{k_3}/w_3} (q_2, \sigma_2\beta_2) \xrightarrow{u_4^{k_2}/w_4} (q_2, \sigma_2) \\ (i'_2, \bot) \xrightarrow{u_1u_2^{k_1}/w'_1} (p'_2, \sigma'_2) \xrightarrow{u_2^{k_2}/w'_2} (p'_2, \sigma'_2\beta'_2) \xrightarrow{u_2^{k_3}u_3u_4^{k_3}/w'_3} (q'_2, \sigma'_2\beta'_2) \xrightarrow{u_4^{k_2}/w'_4} (q'_2, \sigma'_2) \end{cases}$$

such that (q_1, σ_1) and (q_2, σ_2) are co-accessible with the same input word u_5, and (q'_1, σ'_1) and (q'_2, σ'_2) are co-accessible with the same input word u'_5. Now for all $i \geq 0$, we let

$$\begin{array}{ll} V^{(i)} = v_1(v_2)^{k_1+ik_2+k_3}v_3(v_4)^{k_1+ik_2+k_3} & W^{(i)} = w_1(w_2)^iw_3(w_4)^i \\ V'^{(i)} = v'_1(v'_2)^{k_1+ik_2+k_3}v'_3(v'_4)^{k_1+ik_2+k_3} & W'^{(i)} = w'_1(w'_2)^iw'_3(w'_4)^i \\ D_1(i) = \Delta(V^{(i)}, V'^{(i)}) & D_2(i) = \Delta(W^{(i)}, W'^{(i)}) \end{array}$$

In other words, $D_1(i)$ (resp. $D_2(i)$) is the delay in T_1 (resp. T_2) accumulated on the input word $u_1(u_2)^{k_1+ik_2+k_3}u_3(u_4)^{k_1+ik_2+k_3}$ by the two runs of T_1 (resp. T_2). There is a relation between the words $V^{(i)}$ and $W^{(i)}$. Indeed, since T_1 and T_2 are equivalent and (q_1, σ_1) and (q_2, σ_2) are both co-accessible by the same input word, for all $i \geq 1$, either $V^{(i)}$ is a prefix of $W^{(i)}$ or $W^{(i)}$ is a prefix of $V^{(i)}$. We have a similar relation between $V'^{(i)}$ and $W'^{(i)}$.

We prove in [10] the following intermediate results: (i) there exists $i_0 \geq 0$ such that for all $i, j \geq i_0$ such that $i \neq j$, $D_1(i) \neq D_1(j)$; (ii) for all $i, j \geq 1$, if $D_1(i) \neq D_1(j)$, then $D_2(i) \neq D_2(j)$. The proofs of those results rely on fundamental properties of word combinatorics and a non-trivial case study that depends on how the words $v_1(v_2)^{k_1+ik_2+k_3}v_3(v_4)^{k_1+ik_2+k_3}$ and $w_1(w_2)^iw_3(w_4)^i$ are overlapping. Thanks to (i) and (ii), we clearly get that $D_2(i_0) \neq D_2(i_0 + 1)$, which provides a counter-example for the matched twinning property. ◀

Subsequential transducers have at most one run per input word, so we get the following:

▶ **Corollary 15.** *Subsequentializable* VPTs *are twinned.*

The MTP is not a sufficient condition to be subsequentializable, as shown for instance by Example 12. Therefore the class of transductions defined by transducers which satisfy the MTP is strictly larger than the class of transductions defined by subsequentializable transducers. However, these transductions are in the same complexity class for evaluation, i.e. polynomial space in the height of the input word for a fixed transducer:

▶ **Theorem 16.** *Let T be an* fVPT*. If T is twinned, then the Turing transducer* LcpInTT(T) *runs, on an input stream u, in space complexity quadratic in the height of u.*

We can state more precisely the space complexity of LcpInTT(T) when T is reduced. In this case, it is in $O\left(|Q|^4 \cdot |\Gamma|^{2|Q|^4+2} \cdot (h(u)+1)^2 \cdot M\right)$, where $M = \max\{|v| : (q, a, v, \gamma, q') \in \delta\}$.

Sketch. Like for the HTP, when T satisfies the MTP, also does the reduced VPT returned by the reduction procedure of [8]. We use a pumping technique to show that for any word $u \in \Sigma^*$ on which there is a run of T, we have $\mathsf{out}_{\neq}^{\max}(u) \leq (h(u)+1)q(T)$ for some function q, whenever the MTP holds for T. This is done as follows: any such word can be uniquely decomposed as $u = u_0c_1u_1c_2 \dots c_nu_n$ with $n \leq h(u)$, each u_i is well-nested and each c_i is a call. Then if the u_i are long enough, we can pump them vertically and horizontally without affecting the global delay, by using the property $\Delta(vv', ww') = \Delta(\Delta(v, w).(v', w'))$. Then we can apply Proposition 3. ◀

6 Conclusion and Remarks

This work investigates the streaming evaluation of nested word transductions, and in particular identifies an interesting class of VPT-transductions which subsumes subsequentializable transductions and can still be efficiently evaluated. The following inclusions summarize the relations between the different *classes* of transductions we have studied:

$$\text{BM fVPTs} \subsetneq \text{Subsequentializable VPTs} \subsetneq \text{twinned fVPTs} \subsetneq \text{HBM fVPTs} \subsetneq \text{fVPTs}$$

Moreover, we have shown that BM, twinned and HBM fVPTs are decidable in coNPTime.

Further Directions An important asset of the class of twinned fVPTs w.r.t. the class of subsequentializable VPTs is that it is decidable. It would thus be interesting to determine whether or not the class of subsequentializable VPTs is decidable. In addition, we also plan to extend our techniques to more expressive transducers, such as those recently introduced in [1], which extend VPTs with global variables and are as expressive as MSO-transductions, and can therefore swap or reverse sub-trees. Another line of work concerns the extension of our evaluation procedure, which holds for functional transductions, to finite valued transductions.

Acknowledgements The authors would like to thank Jean-François Raskin and Stijn Vansummeren for their comments on a preliminary version of this work.

References

1 R. Alur and L. D'Antoni. Streaming tree transducers. *CoRR*, abs/1104.2599, 2011.
2 R. Alur and P. Madhusudan. Adding nesting structure to words. *JACM*, 56(3):16:1–16:43, 2009.
3 Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *PODS*, pages 216–227. ACM-Press, 2005.
4 V. Bárány, C. Löding, and O. Serre. Regularity problems for visibly pushdown languages. In *STACS*, pages 420–431, 2006.
5 D. Barbosa, L. Mignet, and P. Veltri. Studying the XML web: Gathering statistics from an xml sample. *World Wide Web*, 8:413–438, 2005.
6 A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM TOSEM*, 20, 2011.
7 M. Benedikt and A. Jeffrey. Efficient and expressive tree filters. In *FSTTCS*, volume 4855 of *LNCS*, pages 461–472. Springer Verlag, 2007.
8 M. Caralp, P.-A. Reynier, and J.-M. Talbot. A polynomial procedure for trimming visibly pushdown automata. Technical Report hal-00606778, HAL, CNRS, France, 2011.
9 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.
10 E. Filiot, O. Gauwin, P.-A. Reynier, and F. Servais. Streamability of Nested Word Transductions. Technical Report inria-00566409, HAL, CNRS, France, 2011.
11 E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of visibly pushdown transducers. In *MFCS*, volume 6281 of *LNCS*, pages 355–367. Springer, 2010.
12 O. Gauwin, J. Niehren, and S. Tison. Earliest query answering for deterministic nested word automata. In *FCT*, volume 5699 of *LNCS*, pages 121–132. Springer, 2009.
13 M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. *Theor. Comput. Sci.*, 380:199–217, July 2007.
14 T. Harju, O. H. Ibarra, J. Karhumaki, and A. Salomaa. Some decision problems concerning semilinearity and commutation. *JCSS*, 65, 2002.

15 C. Konrad and F. Magniez. Validating XML documents in the streaming model with external memory. Technical Report 1012.3311, arXiv, 2010.

16 V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *WWW*, pages 1053–1062. ACM-Press, 2007.

17 O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

18 P. Madhusudan and M. Viswanathan. Query automata for nested words. In *MFCS*, volume 5734 of *LNCS*, pages 561–573. Springer Berlin / Heidelberg, 2009.

19 L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *ICDT*, pages 299–313, 2007.

20 A. Weber and R. Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, 118(2):327–340, 1995.

The update complexity of selection and related problems

Manoj Gupta[1], Yogish Sabharwal[2], and Sandeep Sen[3]

1 **Indian Institute of Technology, New Delhi**
gmanoj@iitd.ernet.in
2 **IBM Research – India, New Delhi**
ysabharwal@in.ibm.com
3 **Indian Institute of Technology, New Delhi**
ssen@iitd.ernet.in

Abstract

We present a framework for computing with input data specified by intervals, representing uncertainty in the values of the input parameters. To compute a solution, the algorithm can query the input parameters that yield more refined estimates in form of sub-intervals and the objective is to minimize the number of queries. The previous approaches address the scenario where every query returns an exact value. Our framework is more general as it can deal with a wider variety of inputs and query responses and we establish interesting relationships between them that have not been investigated previously. Although some of the approaches of the previous restricted models can be adapted to the more general model, we require more sophisticated techniques for the analysis and we also obtain improved algorithms for the previous model.

We address selection problems in the generalized model and show that there exist 2-update competitive algorithms that do not depend on the lengths or distribution of the sub-intervals and hold against the worst case adversary. We also obtain similar bounds on the competitive ratio for the MST problem in graphs.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Uncertain data, Competitive analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.325

1 Introduction

A common scenario in many computational problems is uncertainty about the precise values of one or more parameters. Many different models have been considered in the database community for dealing with uncertain data. In one of the commonly used models, the uncertain parameters are represented by probability distributions (for a comprehensive survey, see[1]). In another model, the uncertain parameters are represented by interval ranges, wherein the parameter may take on any value within the specified interval (see [12]). In this paper, we focus on the latter model. More formally, we consider the model wherein we want to compute a function $f(x_1, x_2 \ldots x_n)$ where some (or all) x_i's are not fully known. The x_i's are typically known to lie in some range (interval). Any assignment of $x_i = x_i'$ consistent with the known range of x_i is a *feasible realization*. The algorithm can make queries about x_i. This problem has been studied before [12, 9]. A common assumption made in the existing literature is that the exact value of x_i is returned by a single query. However, in many applications, a query about x_i may only yield a more refined estimate of the x_i. As a matter of fact, in many such applications, it is not even possible to obtain the exact value of the

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 325–338

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parameter. As an example, consider the case of handling satellite data such as maps. Due to the large amount of data involved, the data is often stored hierarchically at different scales of resolutions. Typically the data is presented at the highest level of resolution. Depending on the area of interest, data may be retrieved for the next level of resolution for a smaller area (zoom in) by performing a query. Now consider a query to find the closest hospital. Based on the highest scale of resolution, the distances to the hospitals can be determined within a certain range of uncertainty. If the closest hospital cannot be resolved at this level, then further queries are required for certain hospitals to determine which amongst them is the closest. These queries proceed down the hierarchical scales of resolution until it is resolved which is the closest hospital.

Let us illustrate this model using the problem of finding minimum when the exact values are not known but each element is associated with a real interval $[\ell_i, r_i]$. Consider the three elements $x_1 = [3, 17], x_2 = [14, 19], x_3 = [15, 20]$. Clearly any of these can be the minimum element as these are mutually overlapping intervals. Suppose a query returns the exact value, then with three queries, we obtain the complete information and the problem is trivially solved. But the interesting question is - are three queries necessary ? Suppose our first query yields that $x_1 = 10$, then clearly we do not need to make any further queries. On the other hand, the query may yield $x_1 = 16$, so that we are forced to make further queries. In a more general situation, where a query may return a sub-interval, we may obtain $x_1 = [8, 16]$ that doesn't yield any useful information about the identity of the minimum element. On the other hand, if the query returns $[8, 10]$, then we can conclude x_1 to be the minimum even though we do not know the exact value of x_1.

It is natural to compare the number of queries made by the algorithm w.r.t. a hypothetical OPT which can be thought of as a non-deterministic strategy that makes the minimum queries for any feasible realization of the input. Moreover, the algorithm must contain a certificate of correctness of the final answer, viz., that no more queries are necessary regardless of the number of unresolved parameters. This also brings up the related verification problem, i.e., given an incompletely specified problem, does it contain sufficient information for a solution to be computed (without further queries).

1.1 Related Previous Work

Kahan [10] described a technique for maintaining data structures for online problems like flight-path collisions using predictive estimates to obtain higher efficiency. The estimates could be used to prune objects that couldn't provably affect the solution and only those *critical* objects were updated that could affect the answer. Kahan's work laid the foundations for later work on *kinetic* data structures but in his paper, he focussed on describing a framework for minimizing updates of critical objects. Kahan compared the efficiency of his data structures with respect to a non-deterministic optimal algorithm, or more specifically, the competitive ratio in the online setting. If our algorithm makes $q_S(n)$ queries for an input S of size n, then it has competitive ratio c [1] iff for some constant $\alpha > 0$,

$$q_S(n) \leq c \cdot OPT(S) + \alpha$$

where OPT may be thought of as a non-deterministic algorithm (coined as *lucky* in [10]) Note that OPT has an unfair advantage in being able to guess the optimal sequence of

[1] So strictly speaking, the algorithm could take exponential time but may have a bounded competitive ratio.

queries and ensure that it can be verified in collusion with an *adversary* controlling the output of the queries.
For instance, if the given intervals are $x_1 = [2, 6], x_2 = [2, 6], x_3 = [2, 6]$, i.e., all of them are identical, OPT may guess the answer to be x_3 and if the query yields $x_3 = 2$, then it is verified. On the other hand, an algorithm has no means of distinguishing between the x_i's. Even use of randomization does not appear to provide any significant advantage in this scenario. Kahan [10] tackled this issue (without acknowledging as much) by changing the problem definition to that of *reporting* **all** *values that are equal to the minimum.*

Khanna and Tan [12] also used the competitive ratio as a measure of efficiency of their algorithms but their parameterization didn't yield $O(1)$ bounds. Their algorithms for selection was related to the *clique number* (maximum clique size) of the input. They compare with Non-deterministic optimal and show that, no on-line algorithm can achieve a better competitive ratio than the clique number.

A somewhat different model was used by Erlebach et al.[9], who showed how to compute an *exact* minimum spanning tree for graph with interval data using minimal number of queries. The final answer is a combinatorial description (in this case a spanning tree) and not necessarily the weight of the spanning tree. Erlebach et al.[9] proved that their algorithm has competitive ratio 2 when the edge weights are initially specified as *open* intervals. One limitation of their result is the critical use of the property of open intervals which is used to weaken the advantage of OPT in guessing and verifying the answer. Their results on constant competitive ratio do not hold for closed or semi-closed intervals.

A recent motivation for this line of work came from caching problems in distributed databases, (Olston and Widom [13]), where local cached copies are used for faster query processing where the cached values are intervals that are guaranteed to contain the actual value called the *master* value. Their work showed trade-off between the number of queries and the precision Δ of the actual answer. This model was further explored in the work of [6, 5] that tackled fundamental problems like median-finding and shortest-paths. They distinguished between the offline (oblivious) and online (adaptive) queries including weighted versions where queries could have varying costs for different intervals. Unlike the previous work, they compared their efficiency with respect to a worst case optimal rather than a non-deterministic input-specific optimal. Therefore their results cannot be compared effectively with the previous work. Other approaches like [2, 11] minimize the worst case deviation from actual values or minimizing queries to get improved estimates of the expected solution when the distribution is known [7, 8].

2 Our contributions

In this paper, we generalize the query model in several directions. We classify models based on the types of the inputs allowed and the return type of the queries. The input may specify a combination of points (P), open intervals (I) and/or closed intervals (C). This leads to 7 variations , namely, O, C, P, OC, OP, CP and OCP. Similarly queries on intervals (open/closed) may yield points (P), open intervals (I) and/or closed intervals (C)[2]. This also leads to seven variations. These models are specified in Figure 1. We denote the models by X-Y where X denotes the type of the input allowed in the input instance and Y denotes the query return types where X and Y can take values from O, C, P, OC, OP, CP and

[2] We can also handle semi-closed intervals but we have avoided further classification as they don't lead to any interesting results.

	O	C	OC	P	OP	CP	OCP
O	Category-1	(Note α)	(Note α)	(Note α)	(Note α)	(Note α)	(Note α)
C	(Note α)	Category-1	(Note α)	(Note α)	(Note α)	(Note α)	(Note α)
OC	(Note α)	(Note α)	Category-1	(Note α)	(Note α)	(Note α)	(Note α)
P	trivial	-	-	-	-	-	-
OP	Category-2	(Note α)	(Note α)	OP-P	OP-OP	(Note α)	(Note α)
CP	(Note α)	Category-2	(Note α)	Category-3	(Note α)	Category-3	(Note α)
OCP	(Note α)	(Note α)	Category-2	Category-3	(Note α)	(Note α)	Category-3

Figure 1 Models for studying uncertain data problems (see note for α below). The allowed input types listed along the rows and the query return types listed along the columns. (The pure input point model is trivial as no queries are required).

OCP (here the literals O, C and P correspond to open intervals, closed intervals and points respectively). Thus for instance OP-P denotes the model wherein the input can consist of open intervals as well as points and the queries can only return points.

(Note α): Although there are 49 models possible, many of them are unnatural as they can lead to a change of the input type after some initial queries. The framework of such models can be covered under the framework of another suitable model. For instance, a problem under the O-P model would convert to OP-P model after a single query and is thus better studied under the OP-P model. Similarly, the OC-C model can be covered under the OC-OC model.

We categorize the valid models into 5 different categories (See Figure 1). The competitive ratios are based on this categorization of the models. *Category-1* corresponds to the models where the input and query return types are only intervals (O-O, C-C, OC-OC models). *Category-2* corresponds to the models where the input may contain points by the queries only return intervals (OP-O, CP-C, OCP-OC models). *Category-3* corresponds to the models where the input may contain closed intervals and the query may return points. The other two categories correspond to the *OP-P* and *OP-OP* models themselves.

Our main results can be summarized as follows

1. We first generalize the models to practical scenarios wherein queries may return sub-intervals as answers rather than exact values. The sub-intervals need not have any properties with respect to lengths or distributions. In other words, with further queries, we obtain increasingly refined estimates of the values until sufficient information has been obtained, i.e., the *verification* problem can be solved. We show that the *witness based approach* used in the previous models can be adapted to the models considered in this paper. More specifically, we establish interesting relationships between the various models (see Figure 2).
2. We study the selection problem of finding the k^{th} smallest value and present update competitive algorithms with different guarantees for the different models for this problem. We also study the update complexity of minimum spanning tree problem under the different models that is closely related to the extremal selection problem (finding the heaviest edge in a cycle – also called the Red rule).
3. We also show that by deviating from the witness based approach studied in prior literature, we can actually obtain improved bounds for the selection problem. These algorithms attain an *additive* overhead from optimal, that is similar to a competitive ratio of unity for some cases and are interesting in their own right.
4. Given that closed intervals have not been successfully handled in prior literature[9] leading to unbounded competitive ratios, is it possible to characterize the problem more precisely? For instance, do we run into the same issues if we allow queries to return

intervals? One approach for addressing issues with closed intervals is to output all the optimal solutions[10]. It can be quite expensive to output all the solutions. Is there an alternate framework that addresses the issues with closed intervals without determining all the solutions.

We show that this problem is a characteristic of models that allow closed intervals in the input and points to be returned in the queries. We extend our models to handle *closed* intervals by using the notion of lexicographically smallest solution (in case multiple solutions exist). This is a natural version in many problems where the initial ordering is important and we will show later that this has the desired effect of limiting non-deterministic guessing powers of OPT.

Another interesting variation could be assigning cost to a query depending on the the precision of the answer given but we have not addressed this version in this paper. There is a growing body of work that addresses the problem of computing exact answer with minimal queries [3, 4] and coping with more generalized queries is an important and fundamental direction of algorithmic research.

Problem	Competitive ratio	Models	Comment	Source
Extremal selection	$OPT+1$	OCP-P	Report all solutions	Kahan [10]
	$OPT+1$	OP-P	Value	this paper
	$2 \cdot OPT$	Category-1,2 & OP-OP		this paper
	$2 \cdot OPT$	Category-3	lex first	this paper
K-selection	$OPT+1$	OCP-P	Report all solutions	Kahan [10]
	$t \cdot OPT$	CP-P	t = clique no.	Khanna-Tan [12]
	$OPT+k$	OP-P	Value, $\leq k \cdot OPT$	this paper
	$2 \cdot OPT$	Category-1	element	this paper
	$2 \cdot (OPT+k)$	OP-OP		this paper
	$2 \cdot OPT$	Category-3	Value, lex first	this paper
MST	$2 \cdot OPT$	OP-P		Erlebach et al.[9]
	$OPT+\mathcal{C}$	OP-P	$\mathcal{C} \leq OPT$ $\mathcal{C}$ = no. of red rule	this paper
	$2 \cdot OPT$	Category-1,2 & OP-OP		this paper
	$2 \cdot OPT$	Category-3	lex first	this paper

Figure 2 Known results in prior literature and our new results.

3 Problem Definition

We consider a problem $\mathcal{P}$ where we are given an instance $P = (C, A)$ that consists of

- an ordered set of data $C = \{c_1, c_2, \ldots, c_n\}$ called a *configuration*; and
- an ordered set of data $A = \{a_1, a_2, \ldots a_n\}$ called *areas of uncertainty* such that $c_i \in a_i \ \forall i$.

The configuration C is not known to us – only the areas of uncertainty, A, are known. As an example consider the problem, $\mathcal{P}$, of finding the index of the minimum element. An example instance is given by $P_{ex} = (C, A)$ where C is the ordered set of points $C = \{3, 7, 10\}$ and A is the ordered set of intervals (areas of uncertainties) $A = \{(2, 6), (5, 8), (9, 11)\}$.

We focus our discussion to problems where the input is Real data. Thus, the configuration consists of points on the Real line $\Re$, and the areas of uncertainty may be intervals on the Real line. The concepts can be extended to higher-dimensional problems.

Verifier: We are also given a *verifier* V for the problem $\mathcal{P}$, that takes as input the areas of uncertainty, A and returns whether a solution of the problem $\mathcal{P}$ can be determined

from A or not. For the example instance, P_{ex}, described above, the verifier would return false as it cannot determine a solution from the given areas of uncertainty. However, if the intervals were $A = \{(2,5),(6,8),(9,11)\}$, then the verifier would return true as clearly the first interval has to contain the minimum.

Order-Invariance: An important characteristic of the problems we study is that the result of the verifier is only dependent on the ordering of the areas of uncertainty. More formally, consider two instances $P = (C, A)$ and $P' = (C', A')$ where $A = \{a_1, a_2, \ldots, a_n\}$ and $A' = \{a'_1, a'_2, \ldots, a'_n\}$ for the same problem $\mathcal{P}$. We say that P and P' are *order-equivalent* if for every pair of indices $i, j \in \{1, 2, \ldots, n\}$, it can be determined that $a_i \leq a_j$ iff it can be determined that $a'_i \leq a'_j$. We say that a problem $\mathcal{P}$ is *order-invariant* if the verifier returns the same value for any two order-equivalent configuration instances. It is easy to verify that the problems such as selection (finding minimum, finding k^{th}-minimum) and minimum spanning tree are order-invariant.

Update operations: We are allowed to perform *update* operations on the areas. Performing an update operation on area a_i results in knowledge of the area to a greater degree of accuracy. More precisely, performing an update operation on a_i in the instance $P = (C, A)$, where $A = \{a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n\}$ results in another instance $P' = (C, A')$, where $A' = \{a_1, a_2, \ldots, a_{i-1}, a'_i, a_{i+1}, \ldots, a_n\}$ such that a'_i is completely contained in a_i. An important characteristic of the models that we consider is that the results of updates on an area are independent of updates on any other area. That is, given a multi-set $S = \{i_1, i_2, \ldots, i_k\}$ of indices of the areas, applying updates on the corresponding areas results in the same instance, irrespective of the sequence in which these updates are applied. We refer to this as the *update independence property*.

Solution: Our goal is to solve the problem $\mathcal{P}$ by performing minimum number of updates, i.e., perform the minimum number of updates that result in an instance for which the verifier returns true. For a problem instance $P = (C, A)$, a *solution*, S, is defined to be a multi-set of indices $\{i_1, i_2, \ldots, i_k\}$ such that performing updates on the areas $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ results in a problem instance $P' = (C, A')$ for which $V(A')$ returns true, i.e., a solution of the problem can be determined from A without performing any more updates. In this case, we say that S solves the problem instance P. Let $\mathcal{S}(P)$ denote the set of all such solutions. An *optimal solution* is a solution, $S \in \mathcal{S}(P)$ such that any other solution in $\mathcal{S}(P)$ has at least as many indices, i.e., $|S| \leq |S'|$ for all solutions, $S' \in \mathcal{S}(P)$. Therefore, an optimal solution corresponds to a smallest set of indices that need to be updated in order to solve the problem.

As mentioned before, the OP-P and the CP-P models have been studied before. We shall show now show that the algorithms for the OP-P model can be generalized for the many other models for problems that are order-invariant. These update competitive algorithms are based on the concept of witness sets. We discuss these concepts in Section 4; these concepts are borrowed from [4] and presented here with modifications suitable to discuss all our models. Then we discuss how to extend these algorithms to other models.

4 The Witness Set Framework

For a problem instance $P = (C, A)$, a set W is said to be a *witness set* of P if for every solution $S \in \mathcal{S}(P)$, $W \cap S \neq \phi$. Thus, no algorithm can solve P without querying any area from W.

Suppose that we have an algorithm, `WALG`, that given any instance $P = (V, A)$ of the problem, finds a witness-set of size at most k. Then there exists a k-update competitive

algorithm for the problem. The algorithm is presented in Figure 3. It simply keeps applying algorithm WALG to find a witness set of size at most k and updates all the areas in the witness set. It keeps doing this until the problem is solved.

```
Algorithm SOLVE( Problem Instance P, Verifier V, Witness Algorithm WALG )
Input: - problem instance P = (C, A),
       - a verifier algorithm V for the given problem,
       - a witness algorithm WALG for the given problem.
Output: k-update competitive solution to problem instance P

Initialize solution S = {};
If ( V(A) returns false ) /* problem instance is not yet solved */
    W = WALG(P);
    Update the areas in W to reduce the problem instance P to P' ;
    S = S ∪ SOLVE(P', V, WALG);
Endif;
Output S;
```

Figure 3 Algorithm to determine k-update competitive solution given witness algorithm

The following lemma shows that the solution returned by this algorithm is k-update competitive. Note that this result is independent of the model under consideration. The witness algorithm and verifier however are dependent on the underlying model.

▶ **Theorem 1.** *The solution returned by the algorithm in Figure 3 is k-update competitive for the problem instance P.*
Proof omitted.

Witness Algorithms For Different Models. Witness algorithms have been proposed for several problems under the OP-P model. The following theorem shows that the same witness algorithms can be used for various other models as well.

▶ **Theorem 2.** *A witness algorithm for a problem under the OP-P model is also a witness algorithm for the same problem under the category-1, category-2 and OP-OP models (i.e., O-O, C-C, OC-OC, OP-O, CP-C, OCP-OC and OP-OP models).*
Proof omitted.

▶ **Corollary 3.** *Algorithm 3 is k-update competitive under the category-1, category-2 and OP-OP models with the same witness algorithms as that for the OP-P model.*
Proof omitted.

We make an important observation here. While the reduction might seem straightforward, it is important to note many of these reductions are only one-way reduction. For instance, we can reuse the witness algorithm for the OP-P model for the OP-O model but not vice-versa. We demonstrate this later for the k-min selection problem, where we show that while it is possible to design a 2-update competitive algorithm under the OP-P model, it is not possible to design an algorithm that is better than k-update competitive under the OP-O model using witness sets.

Another important observation we make is that prior literature has shown that no algorithm can give bounded update complexity guarantees for the selection problem under the CP-P models. However, we have derived constant factor update-competitive algorithms for models involving closed intervals (i.e., the CP-C, C-C, OC-OC and OCP-OC models). This highlights the fact that the problem is not in dealing with closed intervals but rather with the combination of allowing closed intervals in the input and simultaneously allowing queries to return points for such closed intervals.

5 The selection problem

In an instance $P = (C, A)$ of the k-Min problem, $C = \{p_1, p_2, \cdots, p_n\}$ is an ordered set of points in $\Re$, and $A = \{a_1, a_2, \cdots, a_n\}$ is an ordered set of intervals on $\Re$. The nature of the intervals is determined by the model under consideration. The goal is to find the index of the k^{th} smallest element in C.

We denote by l_j and u_j, the lower and upper ends of the interval a_j respectively. To avoid overloading of notations, we will assume that l_j and u_j always refer to the latest known values for the interval ranges, considering all the updates that have already been performed.

5.1 1-Min

In this section we look at the special case when $k = 1$, i.e., we are interested in finding the index of the smallest value interval.

Witness Algorithm And Verifier. We first present the witness algorithm for the OP-P model. Consider an instance $P = (C, A)$. The witness algorithm chooses the interval with the "smallest l-value" and the along with the interval with the next "smallest l-value" and returns them as the witness set. The verifier simply determines if some interval can be determined to be smaller than all the other intervals. Let $S = \{1..n\}$ denote the set of indices of the intervals. For any subset $S' \subseteq S$, we define $\texttt{order}_l(S')$ to be a permutation of indices in S' in increasing order of the lower values of the corresponding intervals, i.e., $\texttt{order}_l(S') =< j_1, j_2, \cdots, j_m >$, such that $l_{j_1} \leq l_{j_2} \leq \cdots \leq l_{j_m}$. Similarly define $\texttt{order}_u(S') =< j_1, j_2, \cdots, j_m >$, such that $u_{j_1} \leq u_{j_2} \leq \cdots \leq u_{j_m}$.

The witness algorithm and the verifier are formally presented in Figure 4.

`Witness Algorithm:`	`Verifier:`
1. Let $< p_1, p_2, \cdots, p_{\lvert S \rvert} > =$ $\texttt{order}_l(S)$	1. Let $< p_1, p_2, \cdots, p_{\lvert S \rvert} > =$ $\texttt{order}_l(S)$
2. Return a_{p_1} and a_{p_2} as the witness set	2. If $x \leq y$ for all $x \in a_{p_1}$ and $y \in a_{p_j}$, $j \neq 1$, return the interval with index p_1 as the solution Else return false

Figure 4 Witness Algorithm and Verifier for 1-Min under the OP-P model

Note that an interval is declared to be the smallest interval only when no other interval can contain a smaller value. Therefore the algorithm always outputs the correct interval.

Competitiveness. The following lemma shows that the algorithm is 2-update competitive under the OP-P model.

▶ **Lemma 4.** *The set $W = \{p_1, p_2\}$ returned by the algorithm of Figure 4 is a witness set for the 1-Min problem under the OP-P model.*
Proof omitted.

It follows from Theorem 2 and Corollary 3 that we can derive 2-update competitive algorithms for the category-1, category-2 and OP-OP models.

Tight Example. We now show that the update-competitive bound of 2 is tight for all the models that allow the queries to return intervals, i.e., for the category-1, category-2 and OP-OP models (but not the OP-P model). This is demonstrated by the following example. We are given intervals $A = \{a_0, a_1, a_2, \ldots, a_n\}$ where $a_0 = (1, 5)$ and $a_j = (3, 7)$ for all $1 \leq j \leq n$. We argue that any algorithm can be forced to perform $2n$ queries while the OPT

can determine the interval containing the minimum with only n queries. Let S represent the set of intervals $A \setminus \{a_0\}$, i.e., $S = \{a_1, a_2, \ldots, a_n\}$.

Suppose that the algorithm has already performed $2n-1$ queries. The adversary behaves as follows. For the first $n-1$ queries on a_0 it returns the interval $(1+i\varepsilon, 5)$ in the i^{th} query, where ε is a small value $< 1/(2n)$. For the first $n-1$ queries on intervals from the set S it returns the interval $(6, 7)$. The remaining actions of the adversary are based on whether the algorithm performs n queries on a_0 or whether it queries n intervals from S. Note that in performing $2n-1$ queries, the algorithm must encounter one of these cases. These are considered in the following 2 cases:

- Case 1: The algorithm makes n queries to a_0.
  In this case the adversary continues to return the interval $(1+i\varepsilon, 5)$ for the i^{th} query on a_0 where $i \leq 2n-1$ and it returns the interval $(6, 7)$ for each subsequent interval queried from S. Note that in this case, on performing $2n-1$ queries, the algorithm could not have queried all the intervals from S. Therefore at the end of $2n-1$ queries, as there is overlap between interval a_0 and the unqueried intervals from S, the algorithm is forced to make $2n$ queries. The OPT on the other hand can just query all the intervals in S. The adversary will return the interval $(6, 7)$ for OPT on the remaining intervals. Thus, OPT is able to determine that a_0 contains the minimum element by just performing n queries.
- Case 2: The algorithm makes n queries to intervals in S.
  In this case, the adversary returns $(3, 4)$ for the last (n^{th}) interval queried in S. For any subsequent queries to a_0, the adversary continues to return $(1+i\varepsilon, 5)$ for the i^{th} query. Note that in this case, the adversary performs less than n queries on a_0. Therefore at the end of $2n-1$ queries, as there is overlap between interval a_0 and the last queried intervals from S, the algorithm is forced to make $2n$ queries. The OPT on the other hand can just query all the intervals in a_0. The adversary will return the value $(2, 3)$ for OPT on its n^{th} query to a_0 (recall that in this case the algorithm did not perform n queries on a_0). Thus, OPT is able to determine that a_0 contains the minimum element by just performing n queries.

It is surprising that though this tight example demonstrates that we cannot obtain better than 2-update competitive algorithms for these models, it is possible to obtain a 1-update competitive algorithm for the OP-P model; however, this is obtained by an approach different from the Witness Set framework. This is discussed in more detail in Section 6.

5.2 K-Min

We now generalize the 1-min algorithm presented above to the k^{th}-min problem, but under the O-O model. We later discuss issues related to handling points under the OP-P model.

Witness Algorithm And Verifier. We now present a witness algorithm and verifier for this problem under the O-O model.

We say intervals a_i and a_j are disjoint if $\forall x \in a_i, y \in a_j$, $x \leq y$ or vice-verse. The witness algorithm checks if the first $k-1$ interval are *disjoint* with the last $n-k+1$ interval. If that is the case, it returns the witness set of the 1-Min algorithm. Else it chooses a_{p_k} and an interval from S' with largest u value(a_{q_1}) as the witness set.

The $verifier$ takes the first $k-1$ intervals(S') depending on their l values. The $verifier$ checks if these $k-1$ intervals are *disjoint* from the a_{p_k}. Then it takes the last $n-k$ intervals$(S \setminus (S' \cup a_{p_k}))$ and checks if all of them disjoint with a_{p_k}. If both the condition holds, it returns a_{p_k} else it returns false.

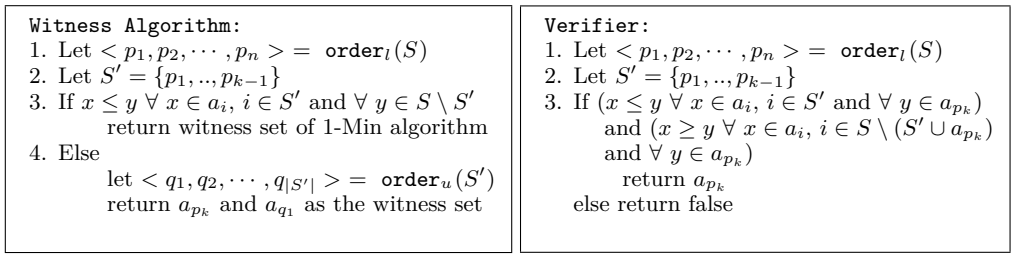


Figure 5 Witness and Verifier Algorithm for K-Min under the O-O model

Competitiveness. The following lemma shows that the algorithm is 2-update competitive for the O-O model. It follows using proofs similar to Theorem 2 and Corollary 3 that we can derive 2-update competitive algorithms for the other category-1 models.

▶ **Lemma 5.** *The witness set W returned by the algorithm of Figure 5 is a witness set for the k-Min problem under the O-O model.*
Proof omitted.

Tight Example. It is not difficult to construct examples similar to that discussed for the 1-Min algorithm to show that the update-competitive bound of 2 is tight under the category-1 models.

It is interesting to note here that while a 2-update competitive algorithm can be designed for the k-min problem under the category-1 models, no algorithm can be better than k-update competitive for this problem under models that allow points, i.e., the category-2 and OP-P models. This is illustrated by the following example[3]. Suppose we have $2k$ areas of which k are open intervals of the form $(0, 5)$ and k are fixed points of the value 3. For the first $k-1$ intervals queried by any algorithm, the adversary returns 1 and for the k^{th} interval, the adversary returns 4 (or interval (3.5,4.5) as the case may be), thereby forcing k queries. However, OPT only needs to update the interval with value 4 and can thereafter return any of the k fixed points of value 3 as the k^{th} smallest.

However, in the next section we show that it is possible to design algorithms for the k-Min problem under these models that allow for points, obtaining update competitive bounds with additive factor k (i.e., the algorithm performs k more updates than OPT). This however is achieved by bypassing the Witness set framework.

6 Bypassing the Witness Set framework

While the witness set framework, studied in prior literature, provides a general method for solving problems with data uncertainty under the update complexity models, it has its limitations. We demonstrate this by presenting algorithms that require to perform only k more queries than OPT for the k^{th}-Min selection problem. Note that, for the 1-Min problem this implies a 1-update competitive algorithm, as only one query more than OPT is required to be performed.

[3] This was pointed out by an anonymous reviewer of a previous version

6.1 1-Min

Consider the following algorithm. We note here that the set of intervals returned by the

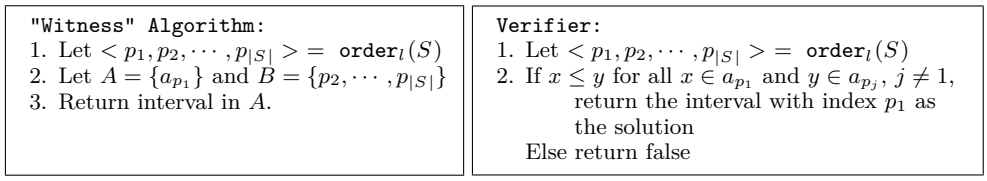


Figure 6 "Witness" Algorithm and Verifier for 1-Min under the OP-P model

"witness" algorithm is not a true witness set. However, we stick to the terminology for the sake of consistency. The algorithm remains the same, it updates the intervals returned by the "witness" algorithm until we obtain a solution.

▶ **Lemma 6.** *Let c_{OPT} be the total number of queries made by OPT to find 1-Min, then total number of queries made by algorithm in Figure 6 is at most $c_{OPT} + 1$ in the OP-P model.*
Proof omitted.

Note that this simple algorithm for 1-Min in OP-P model fails for the OP-O model. Consider the following example. Let there be two intervals I_1= (2,20) and I_2 = (19,21) Suppose at the i^{th} query of I_1, we get a new interval $(d_i, 20)$, where $d_i < 19$, so I_1 and I_2 will always intersect if we just query I_1. The algorithm in Figure 6 always queries I_1, so it takes huge number of queries to find 1-Min. But if we just query I_2, it returns a subinterval (20.5,21). This is what OPT does and uses just one query to find the answer.

6.2 k-Min

Consider the algorithm in Figure 7 for k-selection in the OP-P model which generalizes the result of the algorithm in Figure 6.

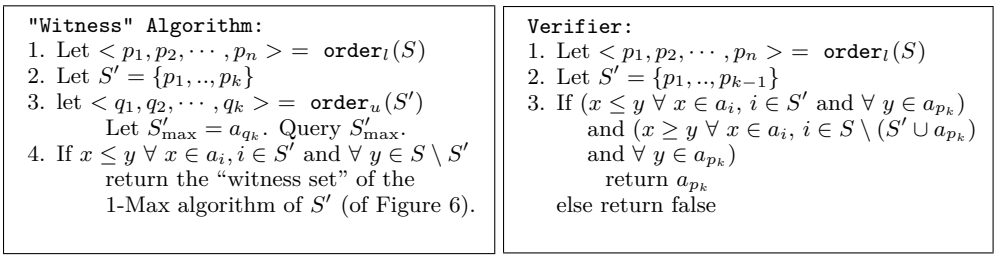


Figure 7 Witness and Verifier Algorithm for K-Min under the OP-P model

▶ **Lemma 7.** *The algorithm of Figure 7 uses atmost $c_{OPT} + \min\{k, n-k\}$ queries where c_{OPT} is the minimum number of queries required by the OPT.*
Proof omitted.

Now let us consider the OP-OP model. Note that since we have $2 \cdot OPT$ algorithms for the OP-O model and an $OPT + k$ algorithm for the OP-P model, we can derive a $2 \cdot (OPT + k)$ algorithm for the OP-OP model by combining these 2 algorithms. This is

done by alternating the witness algorithms of the two models. This ensures that we only need to perform at most twice the number of queries performed by the algorithms of either of the two models.

7 Closed intervals with point returning queries

As discussed above, the competitive ratio is unbounded for the special cases where the input allows for closed intervals and queries may return points (i.e., the category-3 models). For instance consider the problem of finding the index of the minimum element. Further, consider the problem instance $P = (C, A)$ where $a_i = [1, 3]$ for all $1 \leq i \leq n$. The adversary in this case acts as follows; for each of our queries except the last, it returns 2. Finally, for our last query, say on interval a_k, it returns 1. On the other hand, OPT directly queries interval a_k and obtains the optimal solution. This results in an unbounded competitive ratio.

The primary reason for this anomaly is the possibility of existence of multiple optimal solutions. In such cases, the adversary is able to get away with few queries by just querying the necessary intervals that reveal one of the optimal solutions. For any algorithm on the other hand, it is not able to distinguish from the areas of uncertainty (as shown above) which are the necessary intervals to query to reveal the optimal solution.

One of the ways that has been suggested in prior literature to deal with this special case is to require all the optimal solutions to be output. However, it can be quite expensive to output all these solutions. This raises the question of whether other reasonable conditions can be laid on the structure of the required output that are not so expensive but reasonable. We now consider such a condition, which we call the *lexicographic condition*, for which we show that this special can be handled. Recall that the sets C and A that define a problem instance are ordered sets. Thus, the set of indices that define a solution can be considered as a string (called *solution string*) defined as follows: the length of the string is n and the i^{th} element of the string is set to 1 if it defines the solution and 0 otherwise. In the lexicographic setting, amongst all the optimal solutions, we are interested in finding the solution for which the solution string has the smallest lexicographic ordering.

Now consider again the example above. Note that, even though OPT queries a_k and determines a solution with optimal solution value, it cannot terminate without making further queries as it cannot decide whether or not there exists another solution with the same value but a smaller lexicographic ordering.

We note that new witness algorithms may require to be developed for the lexicographic variants of the problems. However, we show by case of examples that these are not very different from the corresponding witness algorithms for the original problems.

It can be shown that once a witness algorithm is developed for a lexicographic variant of the problem under the CP-P model, the same witness algorithm can be extended to other models along the same lines as discussed in Section 4.

Now let us consider the lexicographic variant of the 1-Min problem. In order to obtain the witness algorithm for the lexicographic variant for the category-3 models, the notion of ordering of intervals, $\texttt{order}_l(.)$, needs to be extended to incorporate lexicographic ordering and closed intervals. As before, for any subset $S' \subseteq S$, we define $\texttt{order}_l(S')$ to be a permutation of indices in S' in increasing order of the lower values of the corresponding intervals, i.e., $\texttt{order}_l(S') =< j_1, j_2, \cdots, j_m >$, such that $l_{j_1} \leq l_{j_2} \leq \cdots \leq l_{j_m}$. When comparing two intervals with the same l-values, say l_j and $l_{j'}$, ties are resolved as follows: If a_j contains a point x such that $x < y$ for all $y \in a_{j'}$, then j precedes j' in the ordering;

similarly if $a_{j'}$ contains such a point, then j' precedes j; and if neither can be established, then the lexicographically smaller index precedes the larger one in the ordering. Thus, if one of the intervals, say a_j, is open from the left and another interval, say $a_{j'}$, is either closed from the left or a point, then j' precedes j in the ordering; in all other cases, the lexicographic smaller of j and j' precedes the other in the ordering.

The witness algorithm and verifier are formally presented in Figure 8. Note that the verifier is also modified so that it can check that the minimum interval can be determined or not based on the lexicographic ordering.

`Witness Algorithm:`	`Verifier:`
1. Let $< p_1, p_2, \cdots, p_{\|S\|} > =$ $\texttt{order}_l(S)$ 2. Return a_{p_1} and a_{p_2} as the witness set	1. Let $< p_1, p_2, \cdots, p_{\|S\|} > =$ $\texttt{order}_l(S)$ 2. If $(x \leq y \ \forall\ x \in a_{p_1}$ and $y \in a_{p_j}, p_j > p_1)$ and $(x < y \ \forall\ x \in a_{p_1}$ and $y \in a_{p_j}, p_j < p_1)$, return the interval with index p_1 as the solution Else return false

Figure 8 Witness Algorithm for 1-Min under the CP-P model

The proof of update competitiveness is similar to the case for the original problem.

▶ **Lemma 8.** *The set $W = \{p_1, p_2\}$ returned by the algorithm of Figure 8 is a witness set for the lexicographic* 1*-Min problem under the CP-P model.*
Proof omitted.

The fact that no algorithm can be better than 2-update competitive for the 1-Min problem under the CP-P model follows from the same reasoning as for the OP-P model.

We can extend this 2-update competitive algorithm for the other category-3 models using techniques similar to that in Section 4.

Finally, we can design 2-update competitive algorithms for the k-min version as well under these models by using similar techniques.

8 Minimum Spanning Tree

In the Lexicographic MST problem, we are given a graph $G = (V, E)$. The edge lengths are specified with uncertainty. Let $E = \{e_1, e_2, \ldots, e_n\}$ be the ordered set of edges. Then the ordered set $C = \{v_1, v_2, \cdots, v_n\}$ denotes the values of the edge lengths and the ordered set $A = \{a_1, a_2, \cdots, a_n\}$ denotes the intervals within which the edge lengths are known to lie. The goal is to find the lexicographically smallest MST under the category-3 models.

A 2-update competitive algorithm for the MST problem was given by [9] under the OP-P model. By applying Theorems 2 and Corollary 3, we conclude that it is 2-update competitive for the Category-1,2 and OP-OP models as well. The Lexicographic MST problem can be solved under the Category-3 models with few changes to the algorithm described in [9]. This gives us the following result.

▶ **Theorem 9.** *There exists a 2-update competitive algorithm for the Lexicographic MST problem under the Category-3 models.*

Remark: It may be noted that the algorithm described in [9] in conjunction with Lemma 6 can be used to derive an $OPT + \mathcal{C}$ update competitive algorithm for the MST problem under the OP-OP model where $\mathcal{C}$ is the number of red-rules applied by the optimal algorithm. Note that $\mathcal{C}$ can be much less than OPT.

9 Conclusion

We extended the one-shot query model to the more general situation where a query can return arbitrary sub-intervals as answers and established strong relationships between these models. Many of the previous results in the restricted model can be generalized based on this relationship that simplifies the task of designing algorithms for the more general model. This is far from obvious as the sub-interval query model presents some obvious challenges because the uncertainty (in the values of any parameter) can take an arbitrary number of steps to be resolved and can be controlled by an adversary. One drawback of this approach is that the actual algorithmic complexity is overlooked and we only focus on the competitive ratio which is justified on the basis of very high cost of a query. For future work, the algorithmic complexity needs to be incorporated in a meaningful way.

References

1 Charu C. Aggarwal and Philip S. Yu. A survey of uncertain data algorithms and applications. *IEEE Trans. Knowl. Data Eng.*, 21(5):609–623, 2009.

2 Ionut D. Aron and Pascal Van Hentenryck. On the complexity of the robust spanning tree problem with interval data. *Oper. Res. Lett.*, 32(1):36–40, 2004.

3 Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann 0002, Matús Mihalák, and L. Shankar Ram. Network discovery and verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006.

4 Richard Bruce, Michael Hoffmann, Danny Krizanc, and Rajeev Raman. Efficient update strategies for geometric computing with uncertainty. *Theory Comput. Syst.*, 38(4):411–423, 2005.

5 Tomás Feder, Rajeev Motwani, Liadan O'Callaghan, Chris Olston, and Rina Panigrahy. Computing shortest paths with uncertainty. In *STACS*, pages 367–378, 2003.

6 Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. *SIAM J. Comput.*, 32(2):538–547, 2003.

7 Ashish Goel, Sudipto Guha, and Kamesh Munagala. Asking the right questions: model-driven optimization using probes. In *PODS*, pages 203–212, 2006.

8 Sudipto Guha and Kamesh Munagala. Model-driven optimization using adaptive probes. In *SODA*, pages 308–317, 2007.

9 Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *STACS*, pages 277–288, 2008.

10 Simon Kahan. A model for data in motion. In *STOC*, pages 267–277, 1991.

11 A. Kasperski and P. Zielenski. An approximation algorithm for interval data minmax regret combinatorial optimization problem. *Information Processing Letters*, 97(5):177–180, 2006.

12 Sanjeev Khanna and Wang Chiew Tan. On computing functions with uncertainty. In *PODS*, pages 171–182, 2001.

13 Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB*, pages 144–155, 2000.

A Tight Lower Bound for Streett Complementation*

Yang Cai[1] and Ting Zhang[2]

1 **MIT Computer Science and Artificial Intelligence Laboratory**
The Stata Center, 32-G696, Cambridge, MA 02139 USA
ycai@csail.mit.edu

2 **Department of Computer Science, Iowa State University**
226 Atanasoff Hall, Ames, IA 50011 USA
tingz@iastate.edu

Abstract

Finite automata on infinite words (ω-automata) proved to be a powerful weapon for modeling and reasoning infinite behaviors of reactive systems. Complementation of ω-automata is crucial in many of these applications. But the problem is non-trivial; even after extensive study during the past two decades, we still have an important type of ω-automata, namely Streett automata, for which the gap between the current best lower bound $2^{\Omega(n \lg nk)}$ and upper bound $2^{\Omega(nk \lg nk)}$ is substantial, for the Streett index size k can be exponential in the number of states n. In [4] we showed a construction for complementing Streett automata with the upper bound $2^{O(n \lg n + nk \lg k)}$ for $k = O(n)$ and $2^{O(n^2 \lg n)}$ for $k = \omega(n)$. In this paper we establish a matching lower bound $2^{\Omega(n \lg n + nk \lg k)}$ for $k = O(n)$ and $2^{\Omega(n^2 \lg n)}$ for $k = \omega(n)$, and therefore showing that the construction is asymptotically optimal with respect to the $2^{\Theta(\cdot)}$ notation.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases ω-automata, Streett automata, complementation, lower bounds

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.339

1 Introduction

Complementation is a fundamental notion in automata theory. Given an automaton $\mathcal{A}$, the complementation problem asks to find an automaton $\mathcal{B}$ that accepts exactly all words that $\mathcal{A}$ does not accept. Complementation connects automata theory with mathematical logic due to the natural correspondence between language complementation and logical negation, and hence plays a pivotal role in solving many decision and definability problems in mathematical logic.

A fundamental connection between automata theory and the monadic second order logics was demonstrated by Büchi [1], who started the theory of finite automata on infinite words (ω-automata) [2]. The original ω-automata are now referred to as Büchi automata and Büchi complementation was a key to establish that the class of ω-regular languages (sets of ω-words generated by product $\circ$, union $\cup$, star * and limit $^\omega$) is closed under complementation [2].

Büchi's discovery also has profound repercussions in applied logics. Since the '80s, with increasing demand of reasoning infinite computations of reactive and concurrent systems,

* This research has been supported by NSF CAREER Award CCF-0954132.

ω-automata have been acknowledged as unifying representation for *programs* as well as for *specifications* [26]. Complementation of ω-automata is crucial in many of these applications.

But complementation of ω-automata is non-trivial. Only after extensive studies in the past two decades [23, 16, 18, 6, 27, 20] (also see survey [25]), do we have a good understanding of the complexity of Büchi complementation. But a question about a very important type of ω-automata remains unanswered, namely the complexity of Streett complementation, where the gap between the current lower bound and upper bound is substantial. Streett automata are ones of a kind, because Streett acceptance conditions naturally encode *strong fairness* that infinitely many requests are responded infinitely often, a necessary requirement for meaningful computations [5, 7].

1.1 Related Work

Obtaining nontrivial lower bounds has been difficult. The first nontrivial lower bound for Büchi complementation is $n! \approx (0.36n)^n$, obtained by Michel [16, 15]. In 2006, combining ranking with *full automaton* technique, Yan improved the lower bound of Büchi complementation to $\Omega(L(n))$ [27], which now is matched tightly by the upper bound $O(n^2(L(n))$ [20], where $L(n) \approx (0.76n)^n$. Also established in [27] was a $(\Omega(nk))^n = 2^{\Omega(n \lg nk)}$ tight lower bound (where k is the number of Büchi indices) for generalized Büchi complementation, which also applies to Streett complementation because generalized Büchi automata are a subclass of Streett automata. In [3], we proved a tight lower bound $2^{\Omega(nk \lg n)}$ for Rabin complementation (where Rabin index size k can be as large as $2^{n-\epsilon}$ for any arbitrary but fixed $\epsilon > 0$). Several constructions for Streett complementation exist [24, 9, 19, 14, 17], but all involve at least $2^{O(nk \lg nk)}$ state blow-up, which is significantly higher than the current best lower bound $2^{\Omega(n \lg nk)}$, since the Streett index size k can reach 2^n. Determining the complexity of Streett complementation has been posed as an open problem since the late '80s [24, 14, 27, 25]. In [4] we showed a construction for Streett complementation with the upper bound $2^{O(n \lg n + nk \lg k)}$ for $k = O(n)$ and $2^{O(n^2 \lg n)}$ for $k = \omega(n)$. In this paper we establish a matching lower bound $2^{\Omega(n \lg n + nk \lg k)}$ for $k = O(n)$ and $2^{\Omega(n^2 \lg n)}$ for $k = \omega(n)$, and therefore showing that the construction in [4] is essentially optimal at the granularity of $2^{\Theta(\cdot)}$. This lower bound is obtained by applying two techniques: *fooling set* and *full automaton*.

1.2 Fooling Set

The fooling set technique is a classic way of obtaining lower bounds on nondeterministic finite automata on finite words (NFA). Let Σ be an alphabet and $\mathscr{L} \subseteq \Sigma^*$ a regular language. A set of pairs $P = \{(x_i, y_i) \mid x_i, y_i \in \Sigma^*, 1 \leq i \leq n\}$ is called a *fooling set* for $\mathscr{L}$, if $x_i y_i \in \mathscr{L}$ for $1 \leq i \leq n$ and $x_i y_j \notin \mathscr{L}$ for $1 \leq i, j \leq n$ and $i \neq j$. If $\mathscr{L}$ has a fooling set P, then any NFA accepting $\mathscr{L}$ has at least $|P|$ states [8]. The purpose of a fooling set is to identify runs with dual properties (called fooling runs): fragments of accepting runs of $\mathscr{L}$, when pieced together in certain ways, induce non-accepting runs. By an argument in the style of Pumping Lemma, a small automaton would not be able to distinguish how it arrives at a state, and hence it cannot differentiate between some accepting runs and some non-accepting ones.

In the setting of ω-automata, a similar technique exists, which we refer to as Michel's scheme [16]. A set $P = \{x_i \in \Sigma^* \mid 1 \leq i \leq n\}$ is called a *fooling set* for $\mathscr{L}$, if $(x_i)^\omega \in \mathscr{L}$ for $1 \leq i \leq n$ and $((x_i)^+(y_j)^+)^\omega \subseteq \overline{\mathscr{L}}$ for $1 \leq i, j \leq n$ and $i \neq j$ [16, 15].

1.3 Full Automaton

Sakoda and Sipser introduced the *full automaton* technique [21] (the name was first coined in [27]) and used it to obtain several completeness and lower bound results on transformations involving 2-way finite automata [21]. In particular, they proved a classic result of automata theory: the lower bound of complementing an NFA with n states is 2^n.

To establish lower bounds for complementation, one starts with designing a class of automata $\mathcal{A}_n$ and then a class of words $\mathcal{W}_n$ such that $\mathcal{W}_n$ are not contained in $\mathscr{L}(\mathcal{A}_n)$. Next one shows that runs of purported complementary automata $\mathcal{C}_n$ on $\mathcal{W}_n$ exhibit dual properties by application of the fooling set technique. However, some fooling runs can only be generated by long and sophisticated words, which are very difficult to be "guessed" right from the beginning. The ingenuity of the full automaton technique is to remove two levels of indirections: since the ultimate goal is to construct fooling runs, why should not one start with runs directly, and build $\mathcal{W}_n$ and $\mathcal{A}_n$ later?

Without a priori constraints imposed from $\mathcal{A}_n$ or $\mathcal{W}_n$ (they do not exist yet), full automata operate on all possible runs; for a full automaton of n states, every possible unit transition graph (bipartite graph with $2n$ vertices) is identified with a letter, and words are nothing but potential run graphs. Removing the two levels of indirections proved to be powerful. By this technique, the 2^n lower bound proof for complementing NFA was surprisingly short and easy to understand [21] (a fooling set method was implicit in the proof).

We should note that full automata operate on large alphabets whose size grows exponentially with the state size, but this does not essentially limit its application to automata on conventional alphabets. By an encoding trick, a large alphabet can be mapped to a small alphabet with no compromise to lower bound results [22, 27, 3].

1.4 Ranking

For ω-automata, the power of fooling set and full automaton technique was further enhanced by the use of rankings on run graphs [27, 3]. Since first introduced in [9], rankings have been shown to a powerful tool to represent properties of run graphs; complementation constructions for various types of ω-automata were obtained by discovering respective rankings that precisely characterize those run graphs that contain no accepting path (with respect to source automata) [12, 13, 14, 6, 10]. With the help of rankings, constructing a fooling set amounts to designing certain type of rankings. In fact, as shown below, an explicit description of a fooling set might be very hard to find, but the essential properties the fooling set induce can be concisely represented by certain type of rankings.

1.5 Our Results

In this paper we establish a lower bound $L(n,k)$ for Streett complementation: $2^{\Omega(n\lg n+kn\lg k)}$ for $k=O(n)$ and $2^{\Omega(n^2\lg n)}$ for $k=\omega(n)$, which matches the upper bound obtained in [4]. This lower bound applies to all Streett complementation constructions that output union-closed automata (see Section 2), which include Büchi, generalized Büchi and Streett automata. This bound considerably improves the current best bound $2^{\Omega(n\lg nk)}$ [27], especially in the case $k=\Theta(n)$.

Determinization is another fundamental concept in automata theory and it is closely related to complementation. A deterministic T-automaton can be easily complemented by switching from T-acceptance condition to the dual co-T condition (e.g., Streett vs. Rabin). Therefore, the lower bound $L(n,k)$ also applies to Streett determinization if the output

automata are the dual of union-closed automata. In particular, no construction for Streett determinization can output Rabin automata with state size asymptotically less than $L(n,k)$.

We can get a slightly weaker result for constructions that output Rabin automata (which are not union-closed): no construction for Streett complementation can output Rabin automata with state size $n' \leq L(n,k)$ and index size $k' = O(n')$, due to the fact that a Rabin automaton with state n' and index size k' can be translated to an equivalent Büchi automaton with $O(n'k')$ states. For the same reason, no construction for Streett determinization can output Streett automata with state size $n' \leq L(n,k)$ and index size $k' = O(n')$.

Even with the fooling set and full automaton techniques and the assistance of rankings, a difficulty remains: in the setting of Streett complementation, how large can a fooling set for a complementary automaton be? The challenge is two-fold. One is to implant potentially contradictory properties in each member of a fooling set so that complementary run graphs can be obtained by certain combinations of those members. The other is to avoid correlations between members of a fooling set so that each member has to be memorized by a distinct state in a purported complementary automaton. By exploiting the nature of Streett acceptance conditions, our fooling set is obtained via a type of multi-dimensional rankings, called Q-rankings, and members in the fooling set are called Q-words. To simultaneously accommodate potentially contradictory properties in multi-dimension requires handling nontrivial subtleties. We shall continue this discussion in Section 3 after presenting the definition of Q-rankings.

1.6 Paper Organization

Section 2 presents notations and basic terminology in automata theory. Section 3 introduces full Streett automata, Q-rankings and Q-words, and use them to establish the lower bound. Section 4 concludes with a discussion. Due to space limit, technical proofs are omitted, but they can be found in the full version of this paper at arXiv:1102.2963.

2 Preliminaries

2.1 Basic Notations

Let $\mathbb{N}$ be the set of natural numbers. We write $[i..j]$ for $\{k \in \mathbb{N} \mid i \leq k \leq j\}$, $[i..j)$ for $[i..j-1]$, $[n]$ for $[0..n)$. For an infinite sequence ϱ, we use $\varrho(i)$ to denote the i-th component for $i \in \mathbb{N}$, $\varrho[i..j]$ (resp. $\varrho[i..j)$) to denote the subsequence of ϱ from position i to position j (resp. $j-1$). Similar notations for finite sequences and we use $|\varrho|$ to denote the length of ϱ. We assume readers are familiar with notations in language theory, such as $\alpha \circ \alpha'$, α^*, α^+ and α^ω where α and α' are sequences and α is finite, and similar ones such as $S \circ S'$, S^*, S^+ and S^ω where S is a set of finite sequences and S' is a set of sequences.

2.2 Automata and Runs

A finite (nondeterministic) automaton on infinite words (ω-automaton) is a 5-tuple $\mathcal{A} = \langle \Sigma, S, Q, \Delta, \mathcal{F} \rangle$, where Σ is an alphabet, S is a finite set of states, $Q \subseteq S$ is a set of initial states, $\Delta \subseteq S \times \Sigma \times S$ is a transition relation, and $\mathcal{F}$ is an acceptance condition.

An infinite word (ω-words) over Σ is an infinite sequence of letters in Σ. A *run* ϱ of $\mathcal{A}$ over an ω-word w is an infinite sequence of states in S such that $\varrho(0) \in Q$ and, $\langle \varrho(i), w(i), \varrho(i+1) \rangle \in \Delta$ for $i \in \mathbb{N}$. Finite runs are defined similarly. Let $Inf(\varrho)$ the set of states that occur infinitely many times in ϱ. An automaton accepts w if there exists a run ϱ

over w that satisfies $\mathcal{F}$, which usually is defined as a predicate on $Inf(\varrho)$. We use $\mathscr{L}(\mathcal{A})$ to denote the set of ω-words accepted by $\mathcal{A}$ and $\overline{\mathscr{L}(\mathcal{A})}$ the complement of $\mathscr{L}(\mathcal{A})$.

2.3 Acceptance Conditions and Automata Types

ω-automata are classified according their acceptance conditions. Below we list three types of ω-automata relevant to this paper. Let F be a subset of Q and G, B two functions $I \to 2^Q$ where $I = [1..k]$ is called the *index set*.

- *Büchi*: $\langle F\rangle$: $Inf(\varrho) \cap F \neq \emptyset$.
- *Streett*: $\langle G, B\rangle_I$: $\forall i \in I,\ Inf(\varrho) \cap G(i) \neq \emptyset \to Inf(\varrho) \cap B(i) \neq \emptyset$.
- *Rabin*: $[G, B]_I$: $\exists i \in I,\ Inf(\varrho) \cap G(i) \neq \emptyset \wedge Inf(\varrho) \cap B(i) = \emptyset$.

Note that Streett and Rabin are dual to each other. An automaton $\mathcal{A}$ is called *union-closed* if when two runs ϱ and ϱ' are accepting, so is any run ϱ'' if $Inf(\varrho'') = Inf(\varrho) \cup Inf(\varrho')$. It is easy to verify that both Büchi and Streett automata are union-closed while Rabin automata are not. Let $J \subseteq I$. We use $\langle G, B\rangle_J$ to denote the Streett condition with respect to only indices in J. When J is a singleton, say $J = \{j\}$, we simply write $\langle G(j), B(j)\rangle$ for $\langle G, B\rangle_J$. We can assume that B is injective and the index size k is bound by 2^n, because if $B(i) = B(i')$ for two different $i, i' \in I$, then we can shrink the index set I by replacing $\langle G, B\rangle_{\{i,i'\}}$ by $\langle G(i) \cup G(i'), B(i)\rangle$. The same convention and assumption are used for Rabin condition.

2.4 Δ-Graphs

A Δ-*graph* (run graph) of an ω-word w under $\mathcal{A}$ is a directed graph $\mathscr{G}_w = (V, E)$ where $V = S \times \mathbb{N}$ and $E = \{\langle\langle s, l\rangle, \langle s', l+1\rangle\rangle \in V \times V \mid s, s' \in S,\ l \in \mathbb{N}, \langle s, w(l), s'\rangle \in \Delta\}$. By the *l-th level*, we mean the vertex set $S \times \{l\}$. Let $S = \{s_0, \ldots, s_{n-1}\}$. By *$s_l$-track* we mean the vertex set $\{s_l\} \times \mathbb{N}$. For a subset X of S, we call a vertex $\langle s, l\rangle$ an X-vertex if $s \in X$. We simply use s for $\langle s, l\rangle$ when the index is irrelevant.

A Δ-*graph* $\mathscr{G}_w$ of a finite word w is defined similarly. By $|\mathscr{G}_w|$ we denote the length of $\mathscr{G}_w$, which is the same as $|w|$. $\mathscr{G}_\sigma$ for $\sigma \in \Sigma$ is called a *unit* Δ-graph. A path in $\mathscr{G}_w$ is called a *full path* if the path goes from level 0 to level $|\mathscr{G}_w|$. By $\mathscr{G}_w \circ \mathscr{G}_{w'}$, we mean the concatenation of $\mathscr{G}_w$ and $\mathscr{G}_{w'}$, which is the graph obtained by merging the last level of $\mathscr{G}_w$ with the first level of $\mathscr{G}_{w'}$. Note that $\mathscr{G}_w \circ \mathscr{G}_{w'} = \mathscr{G}_{w \circ w'}$.

Let w be a finite word. For $l, l' \in \mathbb{N}$, $s, s' \in S$ we write $\langle s, l\rangle \xrightarrow{w} \langle s', l'\rangle$ to mean that there exists a run ϱ of $\mathcal{A}$ such that $\varrho[l..l']$, the subsequence $\varrho(l)\varrho(l+1)\cdots\varrho(l')$ of ϱ, is a finite run of $\mathcal{A}$ from s to s' over w. We simply write $s \xrightarrow{w} s'$, when omitting level indices causes no confusion.

2.5 Full Automata

A full automaton $\langle \Sigma, S, Q, \Delta, \mathcal{F}\rangle$ is a finite automaton with the following conditions: $\Sigma = 2^{S \times S}$, $\Delta \subseteq S \times 2^{S \times S} \times S$, and for all $s, s' \in S$, $\sigma \in \Sigma$, $\langle s, \sigma, s'\rangle \in \Delta$ if and only if $\langle s, s'\rangle \in \sigma$ [21, 27, 3]. For full automata, the alphabet Σ and the transition relation Δ are completely determined by S. As stated in the introduction, the essence of full automaton technique is to use run graphs as free as possible, without worrying which word generates which run graph. Let the functional version of Δ be $\delta : \Sigma \to 2^{S \times S}$, where for every $s, s' \in S$ and every $\sigma \in \Sigma$, $\langle s, s'\rangle \in \delta(\sigma)$ if and only if $\langle s, \sigma, s'\rangle \in \Delta$. The function δ maps a letter σ to a unit Δ-graph $\mathscr{G}_\sigma$, which represents the complete behavior of $\mathcal{A}$ over σ (technically speaking, $\mathscr{G}_\sigma$, with index dropped, is the graph of $\delta(\sigma)$). In the setting of full automata, δ is

simply the identity function on $2^{S\times S}$. Words and run graphs are essentially the same thing. From now on we use the two terms interchangeably. For example, for a word w, $s \xrightarrow{w} s'$ is equivalent to say that a full path in $\mathscr{G}_w$ goes from s to s'.

3 Lower Bound

In this section we define full Streett automata, and related Q-rankings and Q-words, and use them to establish the lower bound. From now on, we reserve n and k, respectively, for the effective state size and index size in our construction (except in Theorem 9 and Section 4 where n and k, respectively, mean the state size and index size of a complementation instance). All related notions are in fact parameterized with n and k, but we do not list them explicitly unless required for clarity. Let I be $[1..k]$. We first describe the plan of proof.

For each $k, n > 0$, we define a full Streett automaton $\mathcal{S} = (\Sigma, S, Q, \Delta, \mathcal{F})$ and a set of Q-rankings $f : Q \to [1..n] \times I^k$. For each Q-ranking f, we define a finite Δ-graph $\mathscr{G}_f$, called a Q-word. We then show that for each f, $(\mathscr{G}_f)^\omega \notin \mathscr{L}(\mathcal{S})$, yet $((\mathscr{G}_f)^+(\mathscr{G}_{f'})^+)^\omega \subseteq \mathscr{L}(\mathcal{S})$ for every distinct pair of Q-rankings f and f', that is, Q-words constitute a fooling set for $\overline{\mathscr{L}(\mathcal{S})}$. Using Michel's scheme [16, 15, 27], we show that if a union-closed automaton $\mathcal{C}$ complements $\mathcal{S}$, then its state size is no less than the number of Q-rankings, because otherwise we can "weave" the runs of $(\mathscr{G}_f)^\omega$ and $(\mathscr{G}_{f'})^\omega$ in such a way that $\mathcal{C}$ would accept a word in $((\mathscr{G}_f)^+(\mathscr{G}_{f'})^+)^\omega$, contradicting $((\mathscr{G}_f)^+(\mathscr{G}_{f'})^+)^\omega \subseteq \mathscr{L}(\mathcal{S})$.

▶ **Definition 1** (Full Streett Automata). Let $\{\mathcal{S} = \langle \Sigma, S, Q, \Delta, \mathcal{F}\rangle\}_{n,k>0}$ be a family of full Streett automata such that

1.1 $S = Q \cup P_{\mathrm{G}} \cup P_{\mathrm{B}} \cup T$ where Q, P_{G}, P_{B} and T are pairwise disjoint sets of the following forms: $Q = \{q_0, \cdots, q_{n-1}\}$, $P_{\mathrm{G}} = \{g_1, \cdots, g_k\}$, $T = \{t\}$, and $P_{\mathrm{B}} = \{b_1, \cdots, b_k\}$.
1.2 $\mathcal{F} = \langle G, B\rangle_I$ such that $G(i) = \{g_i\}$ and $B(i) = \{b_i\}$ for $i \in I$.

Q is intended to be the domain of Q-rankings. P_{G} and P_{B} are pools from which singletons $G(i)$'s and $B(i)$'s are formed. T is to be used for building a *bypass* track that makes graph concatenation behaves like a parallel composition so that properties associated with each subgraph are all preserved in the final concatenation.

▶ **Definition 2** (Q-Ranking). A *Q-ranking* for $\mathcal{S}$ is a function $f : Q \to [1..n] \times I^k$, which is identified with a pair of functions $\langle r, h\rangle$, where $r : Q \to [1..n]$ is one-to-one, and $h : Q \to I^k$ maps a state to a permutation of I.

For a Q-ranking $f = \langle r, h\rangle$, we call r (resp. h) the R-ranking or numeric ranking (resp. H-ranking or index ranking) of f. We use *Q-ranks* (resp. *R-ranks*, *H-ranks*) to mean values of Q-rankings (resp. R-rankings, H-rankings). For $q \in Q$, we write $h(q)[i]$ $(i \in I)$ to denote the i-th component of $h(q)$. Let $\mathcal{D}^Q$ be the set of all Q-rankings and $|\mathcal{D}^Q|$ be the size of $\mathcal{D}^Q$. Clearly, we have $n!$ R-rankings and $(k!)^n$ H-rankings, and so $|\mathcal{D}^Q| = (n!)(k!)^n = 2^{\Omega(n \lg n + nk \lg k)}$.

As stated in the introduction, Q-rankings are essential for obtaining the lower bound. It turns out that H-rankings are the core of Q-rankings, for $(k!)^n$ already begins to dominate $n!$ when k is larger than $\lg n$. Now we explain the idea behind the design of H-rankings. Recall that our goal is to have $(\mathscr{G}_f)^\omega \notin \mathscr{L}(\mathcal{S})$ for any Q-ranking f as well as $((\mathscr{G}_f)^+(\mathscr{G}_{f'})^+)^\omega \subseteq \mathscr{L}(\mathcal{S})$ for any two different Q-rankings f and f'. For simplicity, we ignore R-rankings and assume Q-rankings are just H-rankings. We say that a finite path *discharges* obligation j if the path visits $B(j)$ and a finite path *owes* obligation j if the path visits $G(j)$ but does not visit $B(j)$. As shown below, for each $i \in [n]$, q_i-track in $\mathscr{G}_f$ is associated with the k-tuple $f(q_i)$, which is a permutation of I, and exactly k *full* paths in $\mathscr{G}_f$ goes from the beginning of q_i-track to

the end of q_i-track. We say that those paths *on q_i-track*. For each $i \in [n]$ and $j \in I$, the j-th full path on q_i-track owes exactly the obligation $f(q_i)[j]$. Let $\varrho = \varrho_0 \circ \varrho_1 \circ \cdots$ be an infinite path in $(\mathscr{G}_f)^\omega$ where ϱ_t $(t \geq 0)$ is a full path in the t-th $\mathscr{G}_f$. Without R-rankings, our construction prescribes that all ϱ_t start and end at a specific track, say q_i-track, and hence are associated with $f(q_i)$. Obligations associated with all ϱ_t simply form a subset I' of I. However, we impose an ordering $\prec_{f,i}$ on I' (different from the standard numeric ordering) such that $f(q_i)[j] \prec_{f,i} f(q_i)[j']$ if and only if $j < j'$. The ordering $\prec_{f,i}$ is total thanks to $f(q_i)$ being a permutation of I. Then a condition in our construction guarantees that the minimum obligation with respect to $\prec_{f,i}$ will never be discharged on ϱ, and therefore ϱ violates $\langle G, B\rangle_I$. Since this ϱ is chosen arbitrarily, we have $(\mathscr{G}_f)^\omega \notin \mathscr{L}(\mathcal{S})$.

Now let $\mathscr{G} \in ((\mathscr{G}_f)^+(\mathscr{G}_{f'})^+)^\omega$. To show $\mathscr{G} \in \mathscr{L}(\mathcal{S})$, we construct an infinite path $\varrho = \varrho_0 \circ \varrho_1 \circ \cdots$ in $\mathscr{G}$ that satisfies $\langle G, B\rangle_I$, where ϱ_t $(t \geq 0)$ is a full path in the t-th subgraph (which is either $\mathscr{G}_f$ or $\mathscr{G}_{f'}$). Let i be such that $f(q_i) \neq f'(q_i)$ (it is always possible by the assumption $f \neq f'$). Different from before, q_i-track in $\mathscr{G}_f$ is associated with $f(q_i)$ and q_i-track in $\mathscr{G}_{f'}$ is associated with $f'(q_i)$. Since $f(q_i)$ and $f'(q_i)$ are different permutations of I, a condition in our construction ensures that a full path ϱ_f in $\mathscr{G}_f$ and a full path $\varrho_{f'}$ in $\mathscr{G}_{f'}$, both on q_i-track, mutually discharge each other's obligations. So we let all ϱ_t in $\mathscr{G}_f$ be ϱ_f and all ϱ_t in $\mathscr{G}_{f'}$ be $\varrho_{f'}$. Since there are infinitely many ϱ_f and $\varrho_{f'}$ in ϱ, ϱ satisfies $\langle G, B\rangle_I$, giving us $\mathscr{G} \in \mathscr{L}(\mathcal{S})$. Since $\mathscr{G}$ is chosen arbitrarily, we have $((\mathscr{G}_f)^+(\mathscr{G}_{f'})^+)^\omega \subseteq \mathscr{L}(\mathcal{S})$. Now we are read to formally define Q-words.

▶ **Definition 3** (Q-Word)**.** A finite Δ-graph $\mathscr{G}$ is called a *Q-word* if every level of $\mathscr{G}$ is ranked by the same Q-ranking $f = \langle r, h\rangle$ and $\mathscr{G}$ satisfies the following additional conditions.

3.1 For every $q, q' \in Q$, if $r(q) > r(q')$, there exists a full path ϱ from $\langle q, 0\rangle$ to $\langle q', |\mathscr{G}|\rangle$ such that ϱ visits all of $B(1), \ldots, B(k)$.
3.2 For every $q \in Q$, there exist *exactly* k full paths $\varrho_1, \ldots, \varrho_k$ from $\langle q, 0\rangle$ to $\langle q, |\mathscr{G}|\rangle$ such that for every $i \in I$, ϱ_i does not visit $B(h(q)[j])$ for $j \leq i$, but visits $B(h(q)[j])$ for $i < j$, and ϱ_i does not visit $G(h(q)[j])$ for $j < i$, but visits $G(h(q)[i])$.
3.3 Only Q-vertices have outgoing edges at the first level and incoming edges at the last level.
3.4 For every $q, q' \in Q$, there exists no full path from $\langle q, 0\rangle$ to $\langle q', |\mathscr{G}|\rangle$ if $r(q) < r(q')$.

Property (3.1) concerns with only R-rankings. It says that for every two tracks with different R-ranks, a path exists that goes from the track with higher rank to the track with the lower rank, and such a path discharges all obligations in I. So if those (finite) paths occur infinitely often as fragments of an infinite path ϱ, then ϱ clearly satisfies the Streett condition $\langle G, B\rangle_I$. Property (3.2) concerns with only H-rankings. It says that exactly k full *"parallel"* paths exist between the two ends of every track, and each owes exactly one distinct obligation in I. As shown in Theorem 9, Property (3.2) is the core of the whole construction and proof, because with k increasing, H-rankings contribute more and more to the overall complexity. Properties (3.3) and (3.4) are merely technical; they ensure that no other *full paths* exist besides those prescribed by Properties (3.1) and (3.2). Note that in general more than one Q-word could exist for a Q-ranking f. We simply pick an arbitrary one and call it the Q-word of f, denoted by $\mathscr{G}_f$.

▶ **Theorem 4** (Q-Word)**.** *A Q-word exists for every Q-ranking.*

▶ **Example 5** (Q-Word)**.** Let us consider a full Streett automaton $\mathcal{S}$ where $n = 3$, $k = 2$, $Q = \{q_0, q_1, q_2\}$, $T = \{t\}$, $P_{\mathrm{B}} = \{b_1, b_2\}$, $P_{\mathrm{G}} = \{g_1, g_2\}$, and the following Q-ranking $f = \langle r, h\rangle$: $r(q_0) = 2$, $r(q_1) = 1$, $r(q_2) = 3$, $h(q_0) = \langle 1, 2\rangle$, $h(q_1) = \langle 1, 2\rangle$, $h(q_2) = \langle 2, 1\rangle$. Figure 1 shows a Q-word $\mathscr{G}_f$, which consists of two subgraphs $\mathscr{G}_r$ and $\mathscr{G}_h$, where $\mathscr{G}_r$ in turn

consists of two parts: $\mathscr{G}_r^{(1)}$ (level 0 to level 3) and $\mathscr{G}_r^{(2)}$ (level 3 to 6), and $\mathscr{G}_h$ in turn consists of three parts: $\mathscr{G}_h^{(0)}$ (level 6 to level 12), $\mathscr{G}_h^{(1)}$ (level 12 to level 18), and $\mathscr{G}_h^{(2)}$ (level 18 to level 24). $\mathscr{G}_r$ and $\mathscr{G}_h$ are aimed to satisfy Properties (3.1) and (3.2), respectively.

The R-rank (numeric rank) of every level of $\mathscr{G}_r$ is $(2, 1, 3)$. In $\mathscr{G}_r^{(1)}$, a full path ϱ_r starts from $\langle q_2, 0\rangle$ whose R-rank is the highest. The path visits $\langle b_1, 1\rangle$, $\langle b_2, 2\rangle$ and then $\langle q_0, 3\rangle$ whose R-rank is one less than that of q_2. Similarly in $\mathscr{G}_r^{(2)}$, the path continues from $\langle q_2, 3\rangle$, visits $\langle b_1, 4\rangle$, $\langle b_2, 5\rangle$ and ends at $\langle q_1, 6\rangle$ whose R-rank is one less than that of q_0.

The H-rank (index rank) of every level of $\mathscr{G}_h$ is $(\langle 1, 2\rangle, \langle 1, 2\rangle, \langle 2, 1\rangle)$. Let us take a look at $\mathscr{G}_h^{(1)}$. A full path ϱ_h (marked green except the last edge) starts at $\langle q_1, 12\rangle$, visits $\langle b_2, 13\rangle$ and $\langle g_1, 14\rangle$ (because of $h(q_1)[1] = 1$), and enters t-track (the bypass track $\{t\} \times \mathbb{N}$) at $\langle t, 15\rangle$, from where it stays on t-track till reaching $\langle t, 17\rangle$. Another full path ϱ'_h (marked red except the last edge) starts at $\langle q_1, 12\rangle$ too, takes q_1-track to $\langle q_1, 15\rangle$, and then visits $\langle g_2, 16\rangle$ (because of $h(q_1)[2] = 2$), and enters t-track at $\langle t, 17\rangle$. Both ϱ_h and ϱ'_h return to q_1-track at $\langle q_1, 18\rangle$ using the edge $\langle\langle t, 17\rangle, \langle q_1, 18\rangle\rangle$ (marked blue). By $\varrho_{0\to 6}$, $\varrho_{6\to 12}$ and $\varrho_{18\to 24}$ (all marked blue) we denote the q_1-tracks in $\mathscr{G}_r$, in $\mathscr{G}_h^{(0)}$ and in $\mathscr{G}_h^{(2)}$, respectively. It is easy to verify that Property (3.1) with respect to q_2 and q_1 is satisfied by both $\varrho_r \circ \varrho_{6\to 12} \circ \varrho_h \circ \varrho_{18\to 24}$ and $\varrho_r \circ \varrho_{6\to 12} \circ \varrho_{h'} \circ \varrho_{18\to 24}$. Also easily seen is that Property (3.2) with respect to q_1 is satisfied by $\varrho_{0\to 6} \circ \varrho_{6\to 12} \circ \varrho_h \circ \varrho_{18\to 24}$ and $\varrho_{0\to 6} \circ \varrho_{6\to 12} \circ \varrho_{h'} \circ \varrho_{18\to 24}$.

We are ready for the lower bound proof. Let $J \subseteq I$. We use $\langle G, B\rangle_J$ to denote the Streett condition with respect to only indices in J. The corresponding Rabin condition $[G, B]_J$ is similarly defined. When J is a singleton, say $J = \{j\}$, we simply write $\langle G(j), B(j)\rangle$ for $\langle G, B\rangle_J$ and $[G(j), B(j)]$ for $[G, B]_J$. Obviously, if an infinite run satisfies $\langle G, B\rangle_J$ (resp. $[G, B]_J$), then the run also satisfies $\langle G, B\rangle_{J'}$ (resp. $[G, B]_{J'}$) for $J' \subseteq J$ (resp. $J \subseteq J' \subseteq I$).

▶ **Lemma 6.** *For every Q-ranking f, $(\mathscr{G}_f)^\omega \notin \mathscr{L}(\mathcal{S})$.*

Proof. Let $f = \langle r, h\rangle$, $\mathscr{G} = (\mathscr{G}_f)^\omega$ and ϱ an infinite path in $\mathscr{G}$. For simplicity, we assume ϱ only lists states appearing on the boundaries of $\mathscr{G}_f$ fragments; for any $j \geq 0$, $\varrho(j)$ (resp. $\varrho(j+1)$) is a state in the first (resp. last) level of the j-th $\mathscr{G}_f$ fragment. Let $\varrho[j, j+1]$ denote the finite fragment from $\varrho(j)$ to $\varrho(j+1)$. Let $\varrho[j, \infty]$ denote the suffix of ϱ beginning from $\varrho(j)$.

By Property (3.3), $\varrho(i) \in Q$ for $i \geq 0$. By Property (3.4), ϱ eventually stabilizes on R-ranks in the sense that there exists a j_0 such that for any $j \geq j_0$, $r(\varrho(j)) = r(\varrho(j+1))$. Because every level of $\mathscr{G}$ has the same rank, ϱ stabilizes on a (horizontal) track after j_0, i.e., there exists $i \in [n]$ such that $\varrho(j) = q_i$ for $j \geq j_0$. Property (3.2) says that there are *exactly k full* paths $\varrho_1, \ldots, \varrho_k$ from $\langle q_i, 0\rangle$ to $\langle q_i, |\mathscr{G}_f|\rangle$ in $\mathscr{G}_f$. Therefore, $\varrho[j_0, \infty]$ can be divided into the infinite sequence $\varrho[j_0, j_0+1], \varrho[j_0+1, j_0+2], \ldots$, each of which is one of $\varrho_1, \ldots, \varrho_k$. Let $k_0 \in I$ be the smallest index such that ϱ_{k_0} appears infinitely often in this sequence, i.e., for some $j_1 \geq j_0$, none of $\varrho_1, \ldots, \varrho_{k_0-1}$ appears in $\varrho[j_1, \infty]$. By Property (3.2) again, $\varrho[j_1, \infty]$ visits none of $B(h(q_i)[1]), \ldots, B(h(q_i)[k_0])$, but visits $G(h(q_i)[k_0])$ infinitely often (because ϱ_{k_0} appears infinitely often). In particular, ϱ satisfies $[G(t), B(t)]$ for $t = h(q_i)[k_0]$ and hence $[G, B]_I$. Because ϱ is chosen arbitrarily, we have $\mathscr{G} \notin \mathscr{L}(\mathcal{S})$. ◀

▶ **Lemma 7.** *For every two different Q-rankings f and f', $((\mathscr{G}_f)^+ \circ (\mathscr{G}_{f'})^+)^\omega \subseteq \mathscr{L}(\mathcal{S})$.*

Proof. Let $\mathscr{G} \in ((\mathscr{G}_f)^+ \circ (\mathscr{G}_{f'})^+)^\omega$ be an ω-word where both $\mathscr{G}_f$ and $\mathscr{G}_{f'}$ occur infinitely often in $\mathscr{G}$. Let $f = \langle r, h\rangle$ and $f' = \langle r', h'\rangle$. We have two cases: either $r \neq r'$ or $h \neq h'$.

If $r \neq r'$. Since both r and r' are one-to-one functions from Q to $[1..n]$, there must be $i, j \in [n]$ such that $r(q_i) > r(q_j)$ and $r'(q_j) > r'(q_i)$. By Property (3.1), $\mathscr{G}_f$ contains a full path $\varrho_{i\to j}$ from $\langle q_i, 0\rangle$ to $\langle q_j, |\mathscr{G}_f|\rangle$ that visits all of $B(1), \ldots, B(k)$. By the same property,

$\mathscr{G}_{f'}$ contains a path $\varrho'_{j\to i}$ from $\langle q_j, 0\rangle$ to $\langle q_i, |\mathscr{G}_{f'}|\rangle$ that also visits all of $B(1), \ldots, B(k)$. Then $\varrho_{i\to j} \circ \varrho'_{j\to i}$ is a path in $\mathscr{G}_f \circ \mathscr{G}_{f'}$ that visits all of $B(1), \ldots, B(k)$. Also by Property (3.2), $\mathscr{G}_f$ (resp. $\mathscr{G}_{f'}$) contains a path $\varrho_{i\to i}$ (resp. $\varrho'_{i\to i}$) from $\langle q_i, 0\rangle$ to $\langle q_i, |\mathscr{G}_f|\rangle$ (resp. from $\langle q_i, 0\rangle$ to $\langle q_i, |\mathscr{G}_{f'}|\rangle$).

Now we define an infinite path $\hat{\varrho}$ in $\mathscr{G}$ as follows. We pick the finite path $\varrho_{i\to i}$ in every $\mathscr{G}_f$ fragment and $\varrho'_{i\to i}$ in every $\mathscr{G}_{f'}$ fragment, except that in the case where a $\mathscr{G}_f$ fragment is followed immediately by a $\mathscr{G}_{f'}$ fragment, we pick $\varrho_{i\to j}$ in the preceding $\mathscr{G}_f$ and $\varrho'_{j\to i}$ in the following $\mathscr{G}_{f'}$. It is easily seen that $\hat{\varrho}$, in the form $((\varrho_{i\to i})^* \circ (\varrho_{i\to j} \circ \varrho'_{j\to i})^+ \circ (\varrho'_{i\to i})^*)^\omega$, visits all of $B(1), \ldots, B(k)$ infinitely often, and hence it satisfies the Streett condition $\langle G, B\rangle_I$.

If $h \neq h'$. Then there exist $i \in [n]$, $j \in I$ such that $h(q_i)[j] \neq h'(q_i)[j]$ and $h(q_i)[j^*] = h'(q_i)[j^*]$ for $j^* \in [1..j-1]$. Since both $h(q_i)$ and $h'(q_i)$ are permutations of I, we have $j < k$ and $\{\, h(q_i)[j^*] \mid j^* \in [j..k] \,\} = \{\, h'(q_i)[j^*] \mid j^* \in [j..k] \,\}$. By Property (3.2), in $\mathscr{G}_f$ there exists a path $\varrho_{i\to i}$ from $\langle q_i, 0\rangle$ to $\langle q_i, |\mathscr{G}_f|\rangle$ that visits none of $G(h(q_i)[j^*])$ for $j^* \in [1..j-1]$, but visits all of $B(h(q_i)[j^*])$ for $j^* \in [j+1..k]$. Similarly, in $\mathscr{G}_{f'}$ there exists a path $\varrho'_{i\to i}$ from $\langle q_i, 0\rangle$ to $\langle q_i, |\mathscr{G}_{f'}|\rangle$ that visits none of $G(h'(q_i)[j^*])$ for $j^* \in [1..j-1]$, but visits all of $B(h'(q_i)[j^*])$ for $j^* \in [j+1..k]$. Because $h(q_i)$ and $h'(q_i)$ are different permutations of I, $h'(q_i)[j] = h(q_i)[j_0]$ for some $j_0 \in [j+1..k]$ and $h(q_i)[j] = h'(q_i)[j_1]$ for some $j_1 \in [j+1..k]$. It follows that both $\{\, h(q_i)[j^*] \mid j^* \in [j..k] \,\}$ and $\{\, h'(q_i)[j^*] \mid j^* \in [j..k] \,\}$ are equal to $\{\, h(q_i)[j^*] \mid j^* \in [j+1..k] \,\} \cup \{\, h'(q_i)[j^*] \mid j^* \in [j+1..k] \,\}$. Therefore $\varrho_{i\to i} \circ \varrho'_{i\to i}$ (in $\mathscr{G}_f \circ \mathscr{G}_{f'}$) visits all of $B(h(q_i)[j^*])$ for $j^* \in [j..k]$.

Now let $\hat{\varrho}$ be defined as follows: $\hat{\varrho}$ takes $\varrho_{i\to i}$ in every $\mathscr{G}_f$ fragment and $\varrho'_{i\to i}$ in every $\mathscr{G}_{f'}$ fragment. That is, $\hat{\varrho}$ takes the following form $((\varrho_{i\to i})^+ \circ (\varrho'_{i\to i})^+)^\omega$. Recall that $h(q_i)[j^*] = h'(q_i)[j^*]$ for $j^* \in [1..j-1]$. It follows that $\hat{\varrho}$ does not visit any of $G(h(q_i)[j^*])$ for $j^* \in [1..j-1]$ because neither $\varrho_{i\to i}$ nor $\varrho'_{i\to i}$ does. Also since both $\mathscr{G}_f$ and $\mathscr{G}_{f'}$ occur infinitely often in $\mathscr{G}$, $\hat{\varrho}$ contains infinitely many $\varrho_{i\to i} \circ \varrho'_{i\to i}$, which implies that $\hat{\varrho}$ visits all of $B(h(q_i)[j^*])$ for $j^* \in [j..k]$ infinitely often. Since $h(q_i)$ is a permutation of I, $\hat{\varrho}$ satisfies $\langle G, B\rangle_I$.

In either case (whether $r \neq r'$ or $h \neq h'$), $\mathscr{G}$ contains a path that satisfies $\langle G, B\rangle_I$, which means $\mathscr{G} \in \mathscr{L}(\mathcal{S})$. Because $\mathscr{G}$ is arbitrarily chosen, we have $((\mathscr{G}_f)^+ \circ (\mathscr{G}_{f'})^+)^\omega \subseteq \mathscr{L}(\mathcal{S})$. ◀

The following lemma is the core of Michel's scheme [16, 15], recast in the setting of full automata with rankings [27, 3]. Recall that $\mathcal{D}^Q$ denotes the set of all Q-rankings and $|\mathcal{D}^Q|$ denotes the cardinality of $\mathcal{D}^Q$.

▶ **Lemma 8.** *A union-closed automaton that complements $\mathcal{S}$ must have at least $|\mathcal{D}^Q|$ states.*

Proof. Let $\mathcal{C}$ be a union-closed automaton that complements $\mathcal{S}$. By Lemma 6, for every Q-ranking f, $(\mathscr{G}_f)^\omega \in \mathscr{L}(\mathcal{C})$. Let f, f' be two different Q-rankings and $\mathscr{G}_f$ and $\mathscr{G}_{f'}$ the corresponding Q-words. Let ϱ and ϱ' be the corresponding accepting runs of $(\mathscr{G}_f)^\omega$ and $(\mathscr{G}_{f'})^\omega$, respectively. Also let ϱ_0 and ϱ'_0, respectively, be the accepting runs of $(\mathscr{G}_f)^\omega$ and $(\mathscr{G}_{f'})^\omega$ when we treat $\mathscr{G}_f$ and $\mathscr{G}_{f'}$ as *atomic* letters, that is, ϱ_0 (resp. ϱ'_0) only records states visited at the boundary of $\mathscr{G}_f$ (resp. $\mathscr{G}_{f'}$) and is a subsequence of ϱ (resp. ϱ'). Obviously, $Inf(\varrho_0) \subseteq Inf(\varrho)$, $Inf(\varrho'_0) \subseteq Inf(\varrho')$, $Inf(\varrho_0) \neq \emptyset$ and $Inf(\varrho'_0) \neq \emptyset$. If $Inf(\varrho_0) \cap Inf(\varrho'_0) = \emptyset$ for any pair of f and f', then clearly $\mathcal{C}$ has at least $|\mathcal{D}^Q|$ states because the state set of $\mathcal{C}$ contains $|\mathcal{D}^Q|$ pairwise disjoint nonempty subsets.

Therefore we can assume that $Inf(\varrho_0) \cap Inf(\varrho'_0) \neq \emptyset$ for a fixed pair of f and f'. Let q be a state in $Inf(\varrho_0) \cap Inf(\varrho'_0)$. Because q occurs infinitely often in ϱ, then for some $m > 0$, there exists a path in $(\mathscr{G}_f)^m$ that goes from q to q and visits exactly all states in $Inf(\varrho)$ (or equivalently speaking, $\mathcal{C}$, upon reading the input word $(\mathscr{G}_f)^m$, runs from state q to q, visiting exactly all states in $Inf(\varrho)$ during the run). By $q \xrightarrow[!\, Inf(\varrho)]{(\mathscr{G}_f)^m} q$ we denote the existence of such a

path. Similarly, we have $q \xrightarrow[!\,Inf(\varrho')]{(\mathscr{G}_{f'})^{m'}} q$ for some $m' > 0$. Also we have $q_0 \xrightarrow{(\mathscr{G}_f)^{m_0}} q$ where q_0 is an initial state of $\mathcal{C}$. Now consider the following infinite run ϱ^* in the form

$$q_0 \xrightarrow{(\mathscr{G}_f)^{m_0}} q \xrightarrow[!\,Inf(\varrho)]{(\mathscr{G}_f)^{m}} q \xrightarrow[!\,Inf(\varrho')]{(\mathscr{G}_{f'})^{m'}} q \xrightarrow[!\,Inf(\varrho)]{(\mathscr{G}_f)^{m}} q \xrightarrow[!\,Inf(\varrho')]{(\mathscr{G}_{f'})^{m'}} q \xrightarrow[!\,Inf(\varrho)]{(\mathscr{G}_f)^{m}} q \xrightarrow[!\,Inf(\varrho')]{(\mathscr{G}_{f'})^{m'}} q \cdots$$

which is an accepting run of $\mathcal{C}$ for $(\mathscr{G}_f)^{m_0} \circ ((\mathscr{G}_f)^m \circ (\mathscr{G}_{f'})^{m'})^\omega$ because $Inf(\varrho^*) = Inf(\varrho) \cup Inf(\varrho')$. However, by Lemma 7, $(\mathscr{G}_f)^{m_0} \circ ((\mathscr{G}_f)^m \circ (\mathscr{G}_{f'})^{m'})^\omega \in ((\mathscr{G}_f)^+ \circ (\mathscr{G}_{f'})^+)^\omega \subseteq \mathscr{L}(\mathcal{S})$, a contradiction. ◀

▶ **Theorem 9.** *Streett complementation is in $2^{\Omega(n \lg n + kn \lg k)}$ for $k = O(n)$ and in $2^{\Omega(n^2 \lg n)}$ for $k = \omega(n)$, where n and k are the state size and index size of a complementation instance.*

Proof. Here we switch to use n_0 and k_0, respectively, for the effective state size and index size in our construction $\mathcal{S}$. We have $n = 2k_0 + n_0 + 1$. By Lemma 8, the complementation of $\mathcal{S}$ requires $|\mathcal{D}^Q| = 2^{\Omega(n_0 \lg n_0 + n_0 k_0 \lg k_0)}$ states. If $k_0 \leq k$, we can construct a full Streett automaton $\mathcal{S}'$ with state size n and index size k as follows. $\mathcal{S}'$ is almost identical to $\mathcal{S}$ except that its acceptance condition is defined as $\mathcal{F}' = \langle G', B' \rangle_{I'}$ (for $I' = [1..k]$) such that for $i \in [1..k_0]$, $G'(i) = G(i)$ and $B'(i) = B(i)$ and for $i \in [k_0 + 1, k]$, $G'(i) = B'(i) = \emptyset$. It is easily seen that $\mathcal{S}'$ is equivalent to $\mathcal{S}$ and hence the complementation lower bound for $\mathcal{S}$ also applies to that for $\mathcal{S}'$. Now when $k = O(n)$, we can always find n_0 and k_0 such that $k_0 \leq k$, yet $n_0 = \Omega(n)$ and $k_0 = \Omega(k)$, and hence we have the lower bound $2^{\Omega(n \lg n + kn \lg k)}$. When $k = \omega(n)$, we set $k_0 = n_0$ so that $k_0 \leq k$, $n_0 = \Omega(n)$ and $k_0 = \Omega(n)$, and hence we have the lower bound $2^{\Omega(n^2 \lg n)}$. ◀

4 Concluding Remarks

In this paper we proved a tight lower bound $L(n, k)$ for Streett complementation. We note that we can improve the lower bound by two modifications. First, we allow $G(i)$ (resp. $B(i)$) to be arbitrary subsets of P_G (resp. P_B). Second, we also use multi-dimensional R-rankings; the range of r is a set of k-tuples of integers in $[1..n]$. As a result, both R-ranks and H-ranks are k-tuples of integers where k can be as large as 2^n (the current effective k is bounded by n). These two modifications require much more sophisticated definition of Q-rankings and construction of Q-words, but they have no asymptotic effect on $L(n, k)$. The situation is different from Rabin complementation [3], where Q-rankings are also multi-dimensional (though different terms other than Q-rankings and Q-words were used), and each component in a k-tuple (the value of a Q-ranking) is independent from one another, and hence each can impose an independent behavior on Q-words. Put it in another way, no matter how large the index set is (the maximum size can be 2^n), all dual properties, each of which is parameterized with an index, can be realized in one Q-word. For Streett complementation, the diminishing gain when pushing up k made us realize that with increasing number of Q-rankings, more and more correlations occur between Q-rankings. Exploiting these correlations leads us to the discovery of the corresponding upper bound.

Acknowledgment

We would like to thank anonymous reviewers for many useful comments, and we are grateful to Laurel Tweed and Wanwu Wang for carefully proofreading the paper.

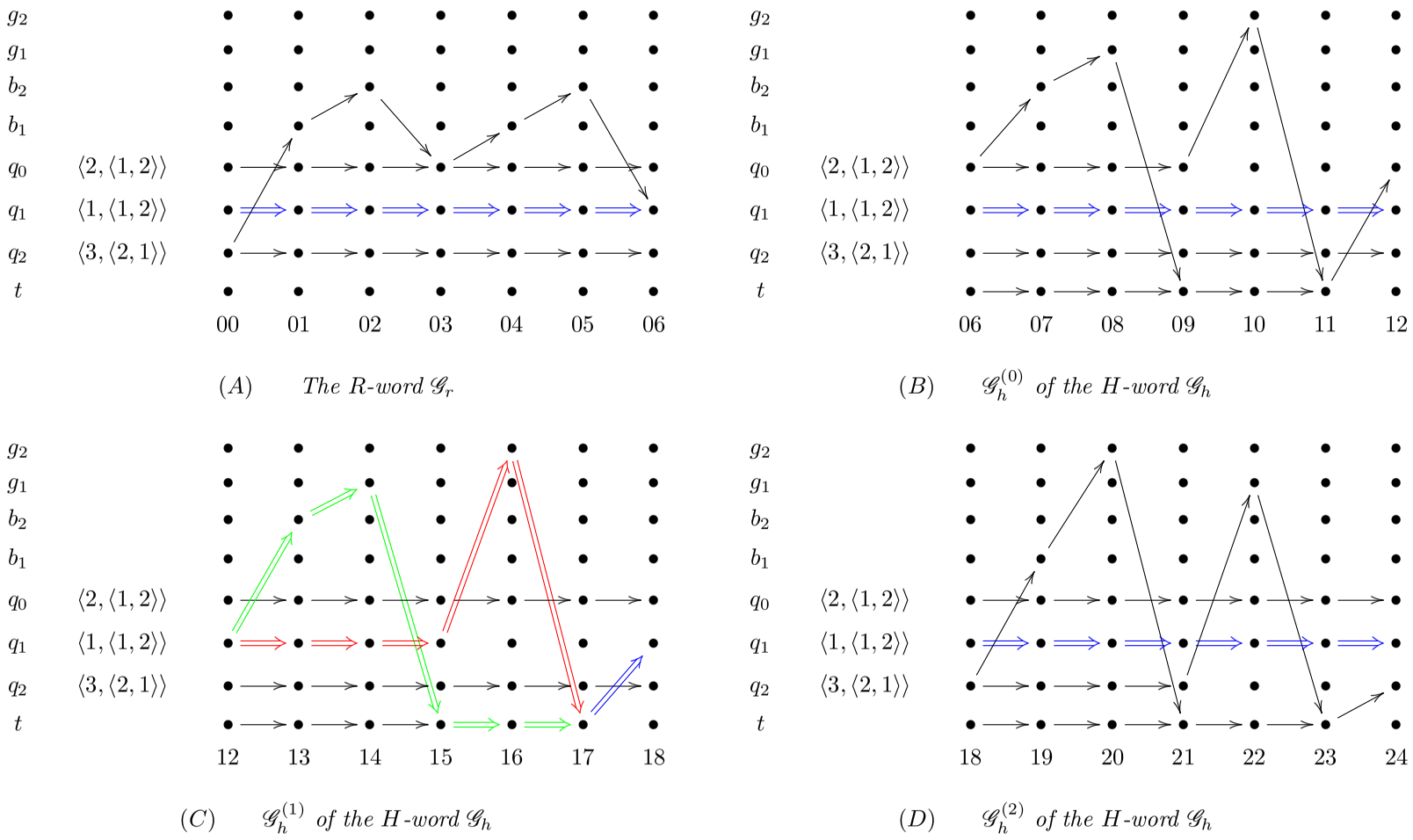


Figure 1 Q-word $\mathscr{G}_f$ $(f = \langle r, h\rangle)$

References

1 J.R. Büchi. Weak Second-order Arithmetic and Finite Automata. *Mathematical Logic Quarterly*, 6(1-6): 66-92, 1960.

2 J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. 1960 Internat. Congr. Logic, Method. and Philos. Sci.*, pp. 1-11, 1966.

3 Y. Cai, T. Zhang, and H. Luo. An improved lower bound for the complementation of Rabin automata. In *Proc. 24th LICS*, pp. 167-176, 2009.

4 Y. Cai and T. Zhang. Tight upper bounds for Streett and parity complementation. In *Proc. 20th CSL*, pp. 112-128, 2011.

5 N. Francez and D. Kozen. Generalized fair termination. In *Proc. 11th POPL*, pp. 46-53, 1984.

6 E. Friedgut and O. Kupferman and M.Y. Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, 17(4): 851-867, 2006.

7 N. Francez. Fairness. *Texts and Monographs in Computer Science*. Springer-Verlag, 1986.

8 Ian Glaister and Jeffrey Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*. 59(2): 75-77, 1996.

9 N. Klarlund. Progress measures for complementation of omega-automata with applications to temporal logic. In *Proc. 32th FOCS*, pp. 358-367, 1991.

10 O. Kupferman. Avoiding Determinization. In *Proc. 21th LICS*, pp. 243-254, 2006.

11 R.P. Kurshan. Computer aided verification of coordinating processes: an automata theoretic approach. Princeton University Press, 1994.

12 O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3): 408-429, 2001.

13 O. Kupferman and M.Y. Vardi. From complementation to certification. In *10th TACAS*, *LNCS 2988*, pp. 591-606, 2004.

14 O. Kupferman and M.Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *Proc. 11th TACAS*, pp. 206-221, 2005.

15 C. Löding. Optimal bounds for transformations of omega-automata. In *Proc. 19th FSTTCS*, *LNCS 1738*, pp. 97-109, 1999.

16 M. Michel. Complementation is more difficult with automata on infinite words. *CNET*, Paris, 1988.

17 N. Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. In *Proc. 21th LICS*, pp. 255-264, 2006.

18 S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pp. 319-327, 1988.

19 S. Safra. Exponential Determinization for ω-Automata with Strong-Fairnes Acceptance Condition. In *Proc. 24th STOC*, pp. 275-327, 1992.

20 S. Schewe. Büchi complementation made tight. In *Proc. 26th STACS*, pp. 661-672, 2009.

21 W. Sakoda, M. Sipser. Nondeterminism and the size of two way finite automata. In *Proc. 10th STOC*, pp. 275-286, 1978.

22 M. Sipser. Lower bounds on the size of sweeping automata. In *Proc. 11th STOC*, pp. 360-364, 1979.

23 A. P. Sistla, M.Y. Vardi, and P.Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217-327, 1987.

24 S. Safra and M.Y. Vardi. On ω-Automata and Temporal Logics. In *Proc. 29th STOC*, pp. 127-137, 1989.

25 M.Y. Vardi. The Büchi complementation saga. In *Proc. 24th STACS*, pp. 12-22, 2007.

26 M.Y. Vardi. and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pp. 332-334, 1986.

27 Q. Yan. Lower bound for complementation of ω-automata via the full automata technique. In *Proc. 33th ICALP*, *LNCS 4052*, pp. 589-600, 2006.

Parameterized Regular Expressions and Their Languages

Pablo Barceló[1], Leonid Libkin[2], and Juan L. Reutter[2]

1 **Department of Computer Science, University of Chile**
pbarcelo@dcc.uchile.cl
2 **School of Informatics, University of Edinburgh**
libkin@inf.ed.ac.uk, juan.reutter@ed.ac.uk

Abstract

We study regular expressions that use variables, or parameters, which are interpreted as alphabet letters. We consider two classes of languages denoted by such expressions: under the possibility semantics, a word belongs to the language if it is denoted by some regular expression obtained by replacing variables with letters; under the certainty semantics, the word must be denoted by every such expression. Such languages are regular, and we show that they naturally arise in several applications such as querying graph databases and program analysis. As the main contribution of the paper, we provide a complete characterization of the complexity of the main computational problems related to such languages: nonemptiness, universality, containment, membership, as well as the problem of constructing NFAs capturing such languages. We also look at the extension when domains of variables could be arbitrary regular languages, and show that under the certainty semantics, languages remain regular and the complexity of the main computational problems does not change.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Regular expressions, complexity, decision problems, regular languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.351

1 Introduction

In this paper we study parameterized regular expressions like $(0x)^*1(xy)^*$ that combine letters from a finite alphabet Σ, such as 0 and 1, and variables, such as x and y. These variables are interpreted as letters from Σ. This gives two ways of defining the language of words over Σ denoted by a parameterized regular expression e. Under the first – possibility – semantics, a word $w \in \Sigma^*$ is in the language $\mathcal{L}_\diamond(e)$ if w is in the language of *some* regular expression e' obtained by substituting alphabet letters for variables. Under the second – certainty – semantics, $w \in \mathcal{L}_\Box(e)$ if w is in the language of *all* regular expressions obtained by substituting alphabet letters for variables. For example, if $e = (0x)^*1(xy)^*$, then $01110 \in \mathcal{L}_\diamond(e)$, as witnessed by the substitution $x \mapsto 1, y \mapsto 0$. The word 1 is in $\mathcal{L}_\Box(e)$, since the starred subexpressions can be replaced by the empty word. As a more involved example of the certainty semantics, the reader can verify that for $e' = (0|1)^*xy(0|1)^*$, the word 10011 is in $\mathcal{L}_\Box(e')$, although no word of length less than 5 can be in $\mathcal{L}_\Box(e')$.

These semantics of parameterized regular expressions arise in a variety of applications, in particular in the fields of querying graph-structured data, and static analysis of programs. We now explain these connections.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 351–362

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Applications in graph databases Graph databases, that describe both data and its topology, have been actively studied over the past few years in connection with such diverse topics as social networks, biological data, semantic Web and RDF, crime detection and analyzing network traffic; see [3] for a survey. The abstract data model is essentially an edge-labeled graph, with edge labels coming from a finite alphabet. This finite alphabet can contain, for example, types of relationships in a social network or a list of RDF properties. In this setting one concentrates on various types of reachability queries, e.g., queries that ask for the existence of a path between nodes with certain properties so that the label of the path forms a word in a given regular language [4, 7, 8, 10]. Note that in this setting of querying topology of a graph database, it is standard to use a finite alphabet for labeling [3].

As in most data management applications, it is common that some information is missing, typically due to using data that is the result of another query or transformation [1, 5, 9]. For example, in a social network we may have edges $a \stackrel{x}{\longmapsto} b$ and $a' \stackrel{x}{\longmapsto} b'$, saying that the relationship between a and b is the same as that between a' and b'. However, the precise nature of such a relationship is unknown, and this is represented by a variable x. Such graphs G whose edges are labeled by letters from Σ and variables from a set $\mathcal{W}$ can be viewed as automata over $\Sigma \cup \mathcal{W}$. In checking the existence of paths between nodes, one normally looks for *certain answers* [16], i.e., answers independent of a particular interpretation of variables.

In the case of graph databases such certain answers can be found as follows. Let a, b be two nodes of G. One can view (G, a, b) as an automaton, with a as the initial state, and b as the final state; its language, over $\Sigma \cup \mathcal{W}$ is given by some regular expression $e(G, a, b)$. Then we can be certain about the existence of a word w from some language L that is the label of a path from a to b iff w also belongs to $\mathcal{L}_\Box(e(G, a, b))$, i.e., iff $L \cap \mathcal{L}_\Box(e(G, a, b))$ is nonempty. Hence, computing $\mathcal{L}_\Box(e)$ is essential for answering queries over graph databases with missing information.

Applications in program analysis That regular expressions with variables appear naturally in program analysis tasks was noticed, for instance, in [20, 21, 23]. One uses the alphabet that consists of symbols related to operations on variables, pointers, or files, e.g., `def` for defining a variable, `use` for using it, `open` for opening a file, or `malloc` for allocating a pointer. A variable then follows: $\texttt{def}(x)$ means defining variable x. While variables and alphabet symbols do not mix freely any more, it is easy to enforce correct syntax with an automaton. An example of a regular condition with parameters is searching for uninitialized variables: $(\neg\texttt{def}(x))^*\texttt{use}(x)$.

Expressions like this are evaluated on a graph that serves as an abstraction of a program. One considers two evaluation problems: whether under some evaluation of variables, either some path, or every path between two nodes satisfies it. This amounts to computing $\mathcal{L}_\Diamond(e)$ and checking whether all paths, or some path between nodes is in that language. In case of uninitialized variables one would be using 'some path' semantics; the need for the 'all paths' semantics arises when one analyzes locking disciplines or constant folding optimizations [20, 23]. So in this case the language of interest for us is $\mathcal{L}_\Diamond(e)$, as one wants to check whether there is an evaluation of variables for which some property of a program is true.

Parameterized regular expressions appeared in other applications as well, e.g., in phase-sequence prediction for dynamic memory allocation [25], or as a compact way to express a family of legal behaviors in hardware verification [6], or as a tool to state regular constraints in constraint satisfaction problems [24].

At the same time, however, very little is known about the basic properties of the languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$. Thus, our main goal is to determine the exact complexity

of the key problems related to languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$. We consider the standard language-theoretic decision problems, such as membership of a word in the language, language nonemptiness, universality, and containment. Since the languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$ are regular, we also consider the complexity of constructing NFAs, over the finite alphabet Σ, that define them.

For all the decision problems, we determine their complexity. In fact, all of them are complete for various complexity classes, from NLogspace to Expspace. We establish upper bounds on the running time of algorithms for constructing NFAs, and then prove matching lower bounds for the sizes of NFAs representing $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$. Finally, we look at extensions where the range of variables need not be just Σ. Under the possibility semantics, such languages need not be regular, but under the certainty semantics, we prove regularity and establish complexity bounds.

Related work There are several related papers on the possibility semantics, notably [11, 14, 18]. Unlike the investigation in this paper, [14, 18] concentrated on the $\mathcal{L}_\Diamond(e)$ semantics in the context of *infinite* alphabets. The motivation of [14] comes from the study of infinite-state systems with finite control (e.g., software with integer parameters). In contrast, for the applications outlined in the introduction, finite alphabets are more appropriate [3, 8, 20, 21]. Results in [14] show that under the possibility semantics and infinite alphabets, the resulting languages can also accepted by non-deterministic register automata [18], and both closure and decidability become problematic. For example, universality and containment are undecidable over infinite alphabets [14]. In contrast, in the classical language-theoretic framework of finite alphabets, closure and decidability are guaranteed, and the key questions are related to the precise complexity of the main decision problems, with most of them requiring new proof techniques.

An analog of the $\mathcal{L}_\Box$ semantics was studied in the context of graph databases in [5]. The model used there is more complex than the simple model of parameterized regular expressions. Essentially, it boils down to automata in which transitions can be labeled with such parameterized expressions, and labels can be shared between different transitions. Motivations for this model come from different ways of incorporating incompleteness into the graph database model. Due to the added complexity, lower bounds for the model of [5] do not extend automatically to parameterized regular expressions, and in the cases when complexity bounds happen to be the same, new proofs are required.

Different forms of succinct representations of regular languages, for instance with squaring, complement, and intersection, are known in the literature, and both decision problems [22] and algorithmic problems [12] have been investigated for them. However, it appears that parameterized regular expressions cannot be used to succinctly define an arbitrary regular expression, nor any arbitrary union or intersection of them. Thus, the study of these expressions requires the development of new tools for understanding the lower bounds of their decision problems.

When we let variables range over words rather than letters, under the possibility semantics $\mathcal{L}_\Diamond$ we may obtain, for example, pattern languages [17] or languages given by expressions with backreferences [2]. These languages need not be regular, and some of the problems (e.g., universality for backreferences) are undecidable [11]. In contrast, we show that under the certainly semantics $\mathcal{L}_\Box$ regularity is preserved, and complexity is similar to the case of variables ranging over letters.

Organization Parameterized regular expressions and their languages are formally defined in Section 2. In Section 3 we define the main problems we study. Complexity of the main decision problems is analyzed in Section 4, and complexity of automata construction in

Section 5. In Section 6 we study extensions when domains of variables need not be single letters.

2 Preliminaries

Let Σ be a finite alphabet, and $\mathcal{V}$ a countably infinite set of variables, disjoint from Σ. Regular expressions over $\Sigma \cup \mathcal{V}$ will be called *parameterized regular expressions*. Regular expressions, as usual, are built from $\emptyset$, the empty word ε, symbols in Σ and $\mathcal{V}$, by operations of concatenation $(\cdot)$, union $(|)$, and the Kleene star $(*)$. Of course each such expression only uses finitely many symbols in $\mathcal{V}$. The size of a regular expression is measured as the total number of symbols needed to write it down (or as the size of its parse tree).

We write $\mathcal{L}(e)$ for the language defined by a regular expression e. If e is a parameterized regular expression that uses variables from a finite set $\mathcal{W} \subset \mathcal{V}$, then $\mathcal{L}(e) \subseteq (\Sigma \cup \mathcal{W})^*$. We are interested in languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$, which are subsets of Σ^*. To define them, we need the notion of a valuation ν which is a mapping from $\mathcal{W}$ to Σ, where $\mathcal{W}$ is the set of variables mentioned in e. By $\nu(e)$ we mean the regular expression over Σ obtained from e by simultaneously replacing each variable $x \in \mathcal{W}$ by $\nu(x)$. For example, if $e = (0x)^*1(xy)^*$ and ν is given by $x \mapsto 1, y \mapsto 0$, then $\nu(e) = (01)^*1(10)^*$.

We now formally define the certainty and possibility semantics for parameterized regular expressions.

▶ **Definition 1** (Acceptance). Let e be a parameterized regular expression. Then:

- $\mathcal{L}_\Box(e) := \bigcap\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation for } e\}$ (certainty semantics)
- $\mathcal{L}_\Diamond(e) := \bigcup\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation for } e\}$ (possibility semantics).

Since each parameterized regular expression uses finitely many variables, the number of possible valuations is finite as well, and thus both $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$ are regular languages over Σ^*.

The usual connection between regular expressions and automata extends to the parameterized case. Each parameterized regular expression e over $\Sigma \cup \mathcal{W}$, where $\mathcal{W}$ is a finite set of variables in $\mathcal{V}$, can of course be translated, in polynomial time, into an NFA $\mathcal{A}_e$ over $\Sigma \cup \mathcal{W}$ such that $\mathcal{L}(\mathcal{A}_e) = \mathcal{L}(e)$. Such equivalences extend to $\mathcal{L}_\Box$ and $\mathcal{L}_\Diamond$. Namely, for an NFA $\mathcal{A}$ over $\Sigma \cup \mathcal{W}$, and a valuation $\nu : \mathcal{W} \to \Sigma$, define $\nu(\mathcal{A})$ as the NFA over Σ that is obtained from $\mathcal{A}$ by replacing each transition of the form (q, x, q') in $\mathcal{A}$ (for q, q' states of $\mathcal{A}$ and $x \in \mathcal{W}$) with the transition $(q, \nu(x), q')$. The following is just an easy observation:

▶ **Lemma 2.** *Let e be a parameterized regular expression, and $\mathcal{A}_e$ be an NFA over $\Sigma \cup \mathcal{V}$ such that $\mathcal{L}(\mathcal{A}_e) = \mathcal{L}(e)$. Then, for every valuation ν, we have $\mathcal{L}(\nu(e)) = \mathcal{L}(\nu(\mathcal{A}_e))$.*

Hence, if we define $\mathcal{L}_\Box(\mathcal{A})$ as $\bigcap_\nu \mathcal{L}(\nu(\mathcal{A}))$, and $\mathcal{L}_\Diamond(\mathcal{A})$ as $\bigcup_\nu \mathcal{L}(\nu(\mathcal{A}))$, then the lemma implies that $\mathcal{L}_\Box(e) = \mathcal{L}_\Box(\mathcal{A}_e)$ and $\mathcal{L}_\Diamond(e) = \mathcal{L}_\Diamond(\mathcal{A}_e)$. Since one can go from regular expressions to NFAs in polynomial time, this will allow us to use both automata and regular expressions interchangeably to establish our results.

3 Basic Problems

We now describe the main problems we study here. For each problem we shall have two versions, depending on which semantics – $\mathcal{L}_\Box$ or $\mathcal{L}_\Diamond$ is used. So each problem will have a subscript $*$ that can be interpreted as $\Box$ or $\Diamond$.

We start with decision problems:

NONEMPTINESS$_*$ Given a parameterized regular expression e, is $\mathcal{L}_*(e) \neq \emptyset$?
MEMBERSHIP$_*$ Given a parameterized regular expression e and a word $w \in \Sigma^*$, is $w \in \mathcal{L}_*(e)$?
UNIVERSALITY$_*$ Given a parameterized regular expression e, is $\mathcal{L}_*(e) = \Sigma^*$?
CONTAINMENT$_*$ Given parameterized regular expressions e_1 and e_2, is $\mathcal{L}_*(e_1) \subseteq \mathcal{L}_*(e_2)$?

A special version of nonemptiness is the problem of intersection with a regular language (used in the database querying example in the introduction):

NONEMPTYINTREG$_*$ Given a parameterized regular expression e, and a regular expression e' over Σ, is $\mathcal{L}(e') \cap \mathcal{L}_*(e) \neq \emptyset$?

The last problem is computational rather than a decision problem:

CONSTRUCTNFA$_*$ Given a parameterized regular expression e, construct an NFA $\mathcal{A}$ over Σ such that $\mathcal{L}_*(e) = \mathcal{L}(\mathcal{A})$.

4 Decision problems

In this section we consider the five decision problems – nonemptiness, membership, universality and containment – and provide precise complexity for them.

Restrictions on regular expressions We shall also consider two restrictions on regular expressions; these will indicate when the problems are inherently very hard or when their complexity can be lowered in some cases. One source of complexity is the repetition of variables in expressions like $(0x)^*1(xy)^*$. When no variable appears more than once in a parameterized regular expression, we call it *simple*. Another source of complexity is infinite languages, so we consider a restriction to expressions of *star-height* 0, in which no Kleene star is used: these denote finite languages, and each finite language is denoted by such an expression.

4.1 Nonemptiness

The problem NONEMPTINESS$_\diamond$ has a trivial solution, since $\mathcal{L}_\diamond(e) \neq \emptyset$ for every parameterized regular expression e (except $e = \emptyset$). So we study this problem for the certainty semantics; for the possibility semantics, we look at the related problem NONEMPTINESS-AUTOMATA$_\diamond$, which, for a given NFA $\mathcal{A}$ over $\Sigma \cup \mathcal{V}$ asks whether $\mathcal{L}_\diamond(\mathcal{A}) \neq \emptyset$.

▶ **Theorem 3.** *The problem* NONEMPTINESS$_\square$ *is* EXPSPACE*-complete.*
The problem NONEMPTINESS-AUTOMATA$_\diamond$ *is* NLOGSPACE*-complete.*

The result for the possibility semantics is by a standard reachability argument. Note that the bound is the same here as in the case of infinite alphabets studied in [14]. To see the upper bound for NONEMPTINESS$_\square$, note that there are exponentially many valuations ν, and each automaton $\nu(\mathcal{A}_e)$ is of polynomial size, so we can use the standard algorithm for checking nonemptiness of the intersection of a family of regular languages which can be solved in polynomial space in terms of the size of its input; since the input to this problem is of exponential size in terms of the original input, the EXPSPACE bound follows. The hardness is by a generic (Turing machine) reduction.

In the proof we use the following property of the certainty semantics, which shows a striking difference with the case of standard regular expressions:

▶ **Lemma 4.** *Given a set $e_1, \ldots, e_k$ of parameterized expressions of size at most $n \geq k$, it is possible to build, in time $O(k \cdot n)$ an expression e' such that $\mathcal{L}_\Box(e')$ is empty if and only if $\mathcal{L}_\Box(e_1) \cap \cdots \cap \mathcal{L}_\Box(e_k)$ is empty.*

The reason the case of the $\mathcal{L}_\Box(e)$ semantics is so different from the usual semantics of regular languages is as follows. It is well known that checking whether the intersection of the languages defined by a finite set S of regular expressions is nonempty is PSPACE-complete [19], and hence under widely held complexity-theoretical assumptions no regular expression r can be constructed in polynomial time from S such that $\mathcal{L}(r)$ is nonempty if and only if $\bigcap_{s \in S} \mathcal{L}(s)$ is nonempty. Lemma 4, on the other hand, says that such a construction is possible for parameterized regular expressions under the certainty semantics.

The generic reduction used in the proof of EXPSPACE-hardness of NONEMPTINESS$_\Box$ also provides lower bounds on the minimal sizes of words in languages $\mathcal{L}_\Box(e)$ (note that the language $\mathcal{L}_\diamond(e)$ always contains a word of the size linear in the size of e).

▶ **Corollary 5.** *There exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a sequence of parameterized regular expressions $\{e_n\}_{n \in \mathbb{N}}$ such that each e_n is of size at most $p(n)$, and every word in the language $\mathcal{L}_\Box(e_n)$ has size at least 2^{2^n}.*

The construction is somewhat involved, but it is easy to see the single-exponential bound (which was hinted at in the first paragraph of the introduction, and which was in fact used in connection with querying incomplete graph data in [5]). For each n, consider an expression $e_n = (0|1)^* x_1 \ldots x_n (0|1)^*$. If a word w is in $\mathcal{L}_\Box(e_n)$, then w must contain every word in $\{0,1\}^n$ as a subword, which implies that its length must be at least $2^n + 1$.

We can also show that the use of Kleene star has a huge impact on complexity, which is not at the same time affected by variable repetitions.

▶ **Proposition 6.** *The problem* NONEMPTINESS$_\Box$ *remains* EXPSPACE*-hard over the class of simple regular expressions, but it is Σ_2^p-complete over the class of expressions of star-height 0.*

4.2 Membership

It is easy to see that MEMBERSHIP$_\Box$ can be solved in coNP, and MEMBERSHIP$_\diamond$ in NP: one just guesses a valuation witnessing $w \in \mathcal{L}(v(e))$ or $w \notin \mathcal{L}(v(e))$. These bounds turn out to be tight.

▶ **Theorem 7.** *The problem* MEMBERSHIP$_\Box$ *is* coNP*-complete.*
The problem MEMBERSHIP$_\diamond$ *is* NP*-complete.*

Proof sketch: We only sketch the proof of NP-hardness (which also works for simple expressions). We use a reduction from 3-SAT. Let $\varphi = \bigwedge_{1 \leq j \leq n} (\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ be a 3-CNF propositional formula over variables $\{p_1, \ldots, p_m\}$. That is, each literal ℓ_j^k, for $1 \leq j \leq n$ and $1 \leq k \leq 3$, is either p_i or $\neg p_i$, for some $i \leq m$. From φ we construct, in polynomial time, a simple parameterized regular expression e over alphabet $\Sigma = \{a, b, c, d, 0, 1\}$ and variables $x_i, \bar{x}_i$, for $1 \leq i \leq m$, and a word w over Σ such that φ is satisfiable if and only if $w \in \mathcal{L}_\diamond(e)$.

The regular expression e is defined as f^*, where $f := a(f_1|g_1|\ldots|f_m|g_m)b$, and the regular expressions f_i and g_i are defined as follows. Intuitively, f_i (resp. g_i) codes the clauses in which p_i occurs positively (resp. negatively). Let $j_1, \ldots, j_r$ enumerate the clauses where the variable p_i appears positively. The expression f_i is defined as

$$f_i \;=\; (c^i \mid d^{j_1} \mid \ \ldots \ \mid d^{j_r}) \cdot x_i.$$

The expression g_i is defined similarly except using indices of clauses where p_i occurs negatively, and the variable $\bar{x}_i$ in place of x_i. Note that e is a simple expression and can be constructed in polynomial time from φ.

The word w is $ac1b\,ac0b\,acc1b\,acc0b\ldots ac^m1b\,ac^m0b\,ad1b\,add1b\ldots ad^n1b$; it too can be constructed in polynomial time from φ. It is now not hard to prove that φ is satisfiable if and only if $w \in \mathcal{L}_\diamond(e)$. □

Note that for the case of the possibility semantics, the bound is the same as for languages over the infinite alphabets [14] (for all problems other than nonemptiness and membership, the bounds will be different). The hardness proof in [14] relies on the infinite size of the alphabet, but one can find an alternative proof that uses only finitely many symbols. Both proofs are by variations of 3-SAT or its complement.

The restrictions to expressions without repetitions, or to finite languages, by themselves do not lower the complexity, but together they make it polynomial.

▶ **Proposition 8.** *The complexity of the membership problem remains as in Theorem 7 over the classes of simple expressions, and expressions of star-height* 0*. Over the class of simple expressions of star-height* 0*,* Membership$_\diamond$ *can be solved in polynomial time (actually, in time* $O(nm\log^2 n)$*, where* n *is the size of the expression and* m *is the size of the word).*

The $\log^2 n$ factor appears due to the complexity of the algorithm for converting regular expressions into ε-free NFAs [15].

Membership for fixed words We next consider a variation of the membership problem: Membership$_*(w)$ asks, for a parameterized regular expression e, whether $w \in \mathcal{L}_*(e)$. In other words, w is fixed. It turns out that for the $\diamond$-semantics, this version is efficiently solvable, but for the $\Box$-semantics, it remains intractable unless restricted to simple expressions.

▶ **Theorem 9.** *There is a word* $w \in \Sigma^*$ *such that the problem* Membership$_\Box(w)$ *is* coNP*-hard (even over the class of expressions of star-height* 0*).*

- *For each word* $w \in \Sigma^*$*, the problem* Membership$_\Box(w)$ *is solvable in linear time, if restricted to the class of simple expressions.*
- *For each word* $w \in \Sigma^*$*, the problem* Membership$_\diamond(w)$ *is solvable in time* $O(n\log^2 n)$*.*

On the other hand, it is straightforward to show that the membership problem for fixed expressions can be solved efficiently for both semantics.

4.3 Universality

Somewhat curiously, the universality problem is more complex for the possibility semantics $\mathcal{L}_\diamond$. Indeed, consider a parameterized expression e over Σ, with variables in $\mathcal{W}$. For the certainty semantics, it suffices to guess a word w and a valuation $\nu : \Sigma \to \mathcal{W}$ such that $w \notin \mathcal{L}(\nu(e))$. This gives a Pspace upper bound for this problem, which is the best that we can do, as the universality problem is Pspace-hard even for complete regular expressions. On the other hand, when solving this problem for the possibility semantics, one can expect that all possible valuations for e will need to be analyzed, which increases the complexity by one exponential. (In fact, when one moves to infinite alphabets, this problem becomes undecidable [14]). The lower bound proof is again by a generic reduction.

▶ **Theorem 10.** *The problem* Universality$_\Box$ *is* Pspace*-complete.*

- *The problem* Universality$_\diamond$ *is* Expspace*-complete.*

Similarly to the nonemptiness problem (studied in Section 4.1), the EXPSPACE bound for UNIVERSALITY$_\diamond$ is quite resilient, as it holds even if for simple expressions (note that it makes no sense to study expressions of star-height 0, as they denote finite languages and thus cannot be universal).

▶ **Proposition 11.** *The problem* UNIVERSALITY$_\diamond$ *remains* EXPSPACE*-hard over the class of simple parameterized regular expressions.*

4.4 Containment

The bounds for the containment problem are easily obtained from the fact that both nonemptiness and universality can be cast as its versions. That is, we have:

▶ **Theorem 12.** *Both* CONTAINMENT$_\square$ *and* CONTAINMENT$_\diamond$ *are* EXPSPACE*-complete.*

Proof: Since $\Sigma^* \subseteq \mathcal{L}_\diamond(e)$ iff UNIVERSALITY$_\diamond(e)$ is true, and $\mathcal{L}_\square(e) \subseteq \emptyset$ iff NONEMPTINESS$_\square(e)$ is false, we get EXPSPACE-hardness for both containment problems. To check whether $\mathcal{L}_\square(e_1) \subseteq \mathcal{L}_\square(e_2)$, we must check that $\bigcap_\nu \mathcal{L}(\nu(e_1)) \cap \overline{\mathcal{L}(\nu'(e_2))} = \emptyset$ for each valuation ν' on e_2. This is doable in EXPSPACE, since one can construct exponentially many automata for $\mathcal{L}(\nu(e_1))$ in EXPTIME, as well as the automaton for the complement $\overline{\mathcal{L}(\nu'(e_2))}$, and checking nonemptiness of the intersection of those is done in polynomial space in terms of their size, i.e., in EXPSPACE. Since this needs to be done for exponentially many valuations ν', the overall EXPSPACE bound follows. The proof for the $\mathcal{L}_\diamond$ semantics is almost identical.

Containment with one fixed expression We look at two variations of the containment problem, when one of the expressions is fixed: CONTAINMENT$_*(e_1, \cdot)$ asks, for a parameterized regular expression e_2, whether $\mathcal{L}_*(e_1) \subseteq \mathcal{L}_*(e_2)$; and CONTAINMENT$_*(\cdot, e_2)$ is defined similarly. The reductions proving Theorem 12 show that CONTAINMENT$_\square(\cdot, e_2)$ and CONTAINMENT$_\diamond(e_1, \cdot)$ remain EXPSPACE-complete. For the other two versions of the problem, the proposition below shows that the complexity is lowered by at least one exponential.

▶ **Proposition 13.** CONTAINMENT$_\square(e_1, \cdot)$ *is* PSPACE*-complete.*
CONTAINMENT$_\diamond(\cdot, e_2)$ *is* coNP*-complete.*

4.5 Intersection with a regular language

This problem is a natural analog of the standard decision problem solved in automata-based verification; we also saw in the introduction that it arises when one computes certain answers to queries over incompletely specified graph databases.

Checking whether $\mathcal{L}(e') \cap \mathcal{L}_\square(e) \neq \emptyset$ can be done in EXPSPACE using the same brute-force algorithm as for the nonemptiness problem (intersection of exponentially many regular languages). Since the nonemptiness problem is a special case with $e' = \Sigma^*$, we get the matching lower bound by Theorem 3. For $\mathcal{L}_\diamond(e)$, an NP upper bound is easy: one just guesses a valuation so that $\mathcal{L}(e') \cap \mathcal{L}(\nu(e)) \neq \emptyset$. If e' denotes a single word w, we have an instance of the membership problem, and hence there is a matching lower bound, by Theorem 7. Summing up, we have:

▶ **Corollary 14.** *The problem* NONEMPTYINTREG$_\square$ *is* EXPSPACE*-complete.*
The problem NONEMPTYINTREG$_\diamond$ *is* NP*-complete.*

5 Computing automata

In this section, we first provide upper bounds for algorithms for building NFAs over Σ capturing $\mathcal{L}_\diamond(e)$ and $\mathcal{L}_\square(e)$, and then prove their optimality, by showing matching lower bounds on the sizes of such NFAs. Recall that we are dealing with the problem ConstructNFA$_*$: Given a parameterized regular expression e, construct an NFA $\mathcal{A}$ over Σ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}_*(e)$.

▶ **Proposition 15.** *The problem* ConstructNFA$_\diamond$ *can be solved in single-exponential time, and the problem* ConstructNFA$_\square$ *can be solved in double-exponential time.*

These bounds are achieved by using naive algorithms for constructing automata: namely, one converts a parameterized regular expression e over variables in a finite set $\mathcal{W}$ into an automaton $\mathcal{A}_e$, and then for $|\Sigma|^{|\mathcal{W}|}$ valuations ν computes the automata $\nu(\mathcal{A}_e)$. This takes exponential time. To obtain an NFA for $\mathcal{L}_\diamond(e)$ one simply combines them with a nondeterministic choice; for $\mathcal{L}_\square(e)$ one takes the product of them, resulting in double-exponential time.

We now show that these complexities are unavoidable, as the smallest NFAs capturing $\mathcal{L}_\diamond(e)$ or $\mathcal{L}_\square(e)$ can be of single or double-exponential size, respectively. We say that the *sizes of minimal NFAs for $\mathcal{L}_*$ are necessarily exponential (resp., double-exponential)* if there exists a family $\{e_n\}_{n\in\mathbb{N}}$ of parameterized regular expressions such that:

- the size of each e_n is $O(n)$, and
- every NFA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A}) = \mathcal{L}_*(e_n)$ has at least 2^n (resp., 2^{2^n}) states.

▶ **Theorem 16.** *The sizes of minimal NFAs are necessarily double-exponential for $\mathcal{L}_\square$, and necessarily exponential for $\mathcal{L}_\diamond$.*

Proof sketch: We begin with the double exponential bound for $\mathcal{L}_\square$. For each $n \in \mathbb{N}$, let e_n be the following parameterized regular expression over alphabet $\Sigma = \{0, 1\}$ and variables $x_1, \ldots, x_{n+1}$:

$$e_n = ((0 \mid 1)^{n+1})^* \cdot x_1 \cdots x_n \cdot x_{n+1} \cdot ((0 \mid 1)^{n+1})^*.$$

Notice that each e_n uses $n + 1$ variables, and is of linear size in n. In order to show that every NFA deciding $\mathcal{L}_\square(e_n)$ has 2^{2^n} states, we use the following result from [13]: if $L \subset \Sigma^*$ is a regular language, and there exists a set of pairs $P = \{(u_i, v_i) \mid 1 \leq i \leq m\} \subseteq \Sigma^* \times \Sigma^*$ such that (1) $u_i v_i \in L$, for every $1 \leq i \leq m$, and (2) $u_j v_i \notin L$, for every $1 \leq i, j \leq m$ and $i \neq j$, then every NFA accepting L has at least m states.

Given a collection S of words over $\{0, 1\}$, let w_S denote the concatenation, in lexicographical order, of all the words that belong to S, and let $w_{\bar{S},n}$ denote the concatenation of all words in $\{0, 1\}^{n+1}$ that are not in S.

Then, define a set of pairs $P_n = \{(w_S, w_{\bar{S},n}) \mid S \subset \{0, 1\}^{n+1} \text{ and } |S| = 2^n\}$. Since there are 2^{n+1} binary words of length $n+1$, there are $\binom{2^{n+1}}{2^n}$ different subsets of $\{0, 1\}^{n+1}$ of size 2^n, and thus P_n contains $\binom{2^{n+1}}{2^n} \geq 2^{2^n}$ pairs. Moreover, one can show that $\mathcal{L}_\square(e_n)$ and P_n satisfy properties (1) and (2) above, which proves the double exponential lower bound.

To show the exponential lower bound for $\mathcal{L}_\diamond$, define $e_n = (x_1 \cdots x_n)^*$, and let $P_n = \{(w, w) \mid w \in \{0, 1\}^n\}$. Clearly, P_n contains 2^n pairs. All that is left to do is to show that $\mathcal{L}_\diamond(e_n)$ and P_n satisfy properties (1) and (2) above. Details are omitted. □

Note that the bounds of Theorem 16 apply to simple regular expressions.

The table in Fig. 1 summarizes the main results in Sections 4 and 5.

Problem \ Semantics	Certainty □	Possibility ◇
NONEMPTINESS	EXPSPACE-complete	NLOGSPACE-complete (for automata)
MEMBERSHIP	coNP-complete	NP-complete
CONTAINMENT	EXPSPACE-complete	EXPSPACE-complete
UNIVERSALITY	PSPACE-complete	EXPSPACE-complete
NONEMPTYINTREG	EXPSPACE-complete	NP-complete
CONSTRUCTNFA	double-exponential	single-exponential

Figure 1 Summary of complexity results

6 Extending domains of variables

So far we assumed that variables take values in Σ: our valuations were partial maps $\nu : \mathcal{V} \to \Sigma$. We now consider a more general case when the range of each variable is a regular subset of Σ^*.

Let e be a parameterized regular expression with variables $x_1, \ldots, x_n$, and let $L_1, \ldots, L_n \subseteq \Sigma^*$ be nonempty regular languages. We shall write $\bar{L}$ for $(L_1, \ldots, L_n)$. A valuation in $\bar{L}$ is a map $\nu : \bar{x} \to \bar{L}$ such that $\nu(x_i) \in L_i$ for each $i \leq n$. Under such a valuation, each parameterized regular expression e is mapped into a usual regular expression $\nu(e)$ over Σ, in which each variable x_i is replaced by the word $\nu(x_i)$. Hence we can still define

$$\begin{aligned} \mathcal{L}_\Box(e;\bar{L}) &= \bigcap\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation over } \bar{L}\} \\ \mathcal{L}_\Diamond(e;\bar{L}) &= \bigcup\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation over } \bar{L}\} \end{aligned}$$

According to this notation, $\mathcal{L}_\Box(e) = \mathcal{L}_\Box(e;(\Sigma,\ldots,\Sigma))$, and likewise for $\mathcal{L}_\Diamond$.

Note however that intersections and unions are now infinite, if some of the languages L_i's are infinite, so we cannot conclude, as before, that we deal with regular languages. And indeed they are not: for example, $\mathcal{L}_\Diamond(xx;\Sigma^*)$ is the set of square words, and thus not regular.

We now consider two cases. If each L_i is a finite language, we show that all the complexity results in Fig. 1 remain true. Then we look at the case of arbitrary regular L_i's. Languages $\mathcal{L}_\Diamond(e;\bar{L})$ need not be regular anymore, but languages $\mathcal{L}_\Box(e;\bar{L})$ still are, and we prove that the complexity bounds from the certainty column of Fig. 1 remain true. For complexity results, we assume that in the input $(e;\bar{L})$, each domain L_i is given either as a regular expression or an NFA over Σ.

6.1 The case of finite domains

If all domain languages L_i's are finite, all the lower bounds apply (they were shown when each $L_i = \Sigma$). For upper bounds, note that each finite L_i contains at most exponentially many words in the size of either a regular expression or an NFA that gives it, and each such word is polynomial size. Thus, the number of valuations is at most exponential in the size of the input, and each valuation can be represented in polynomial time. The following is then straightforward.

▶ **Proposition 17.** *If domains L_i's of all variables are finite nonempty subsets of Σ^*, then both $\mathcal{L}_\Box(e;\bar{L})$ and $\mathcal{L}_\Diamond(e;\bar{L})$ are regular languages, and all the complexity bounds on the problems related to them are exactly the same as stated in Fig. 1.*

6.2 The case of infinite domains

We have already seen that if just one of the domains is infinite, then $\mathcal{L}_\diamond(e;\bar{L})$ need not be regular (the $\mathcal{L}_\diamond(xx;\Sigma^*)$ example). Somewhat surprisingly, however, in the case of the certainty semantics, we recover not only regularity but also all the complexity bounds.

▶ **Theorem 18.** *For each parameterized regular expression e using variables $x_1,\ldots,x_n$ and for each an n-tuple $\bar{L}$ of regular languages over Σ, the language $\mathcal{L}_\square(e;\bar{L}) \subseteq \Sigma^*$ is regular. Moreover, the complexity bounds are exactly the same as in the $\square$ column of the table in Fig. 1.*

Proof sketch: We only need to be concerned about regularity of $\mathcal{L}_\square(e;\bar{L})$ and upper complexity bounds, as the proofs of lower bounds apply for the case when all $L_i = \Sigma$. For this, it suffices to prove that there is a finite set U of NFAs so that $\mathcal{L}_\square(e;\bar{L}) = \bigcap_{\mathcal{A}\in U} \mathcal{L}(\mathcal{A})$. Moreover, it follows from analyzing the proofs of upper complexity bounds, that the complexity results will remain the same if the following can be shown about the set U:

- its size is at most exponential in the size of the input;
- checking whether $\mathcal{A} \in U$ can be done in time polynomial in the size of $\mathcal{A}$;
- each $\mathcal{A} \in U$ is of size polynomial in the size of the input $(e;\bar{L})$.

To show these, take $\mathcal{A}_e$ and from it construct a reduced automaton $\mathcal{A}'_e$ in which all transitions (q, x_i, q') are eliminated whenever L_i is infinite. We then show that $\mathcal{L}_\square(\mathcal{A}_e;\bar{L}) = \mathcal{L}_\square(\mathcal{A}'_e;\bar{L})$ (the definition of $\mathcal{L}_\square$ extends naturally from regular expressions to automata for arbitrary domains). This observation generates a finite set U of NFAs which results from applying valuations with finite codomains to $\mathcal{A}'_e$. It is now possible to show that these automata satisfy the required properties.

7 Future work

For most bounds (except universality and containment), the complexity under the possibility semantics is reasonable, while for the certainty semantics it is quite high (i.e., double-exponential in practice). At the same time, the concept of $\mathcal{L}_\square(e)$ captures many query answering scenarios over graph databases with incomplete information [5]. One of the future directions of this work is to devise better algorithms for problems related to the certainty semantics under restrictions arising in the context of querying graph databases.

Another line of work has to do with closure properties: we know that results of Boolean operations on languages $\mathcal{L}_\square(e)$ and $\mathcal{L}_\diamond(e)$ are regular and can be represented by NFAs; the bounds on sizes of such NFAs follow from the results shown here. However, it is conceivable that such NFAs can be succinctly represented by parameterized regular expressions. To be concrete, one can easily derive from results in Section 5 that there is a doubly-exponential size NFA $\mathcal{A}$ so that $\mathcal{L}(\mathcal{A}) = \mathcal{L}_\square(e_1) \cap \mathcal{L}_\square(e_2)$, and that this bound is optimal. However, it leaves open a possibility that there is a much more succinct parameterized regular expression e so that $\mathcal{L}_\square(e) = \mathcal{L}_\square(e_1) \cap \mathcal{L}_\square(e_2)$; in fact, nothing that we have shown contradicts the existence of a polynomial-size expression with this property. We plan to study bounds on such regular expressions in the future.

Acknowledgment We thank Marian Kędzierski for helpful comments. Partial support provided by Fondecyt grant 1110171, EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599.

References

1 S. Abiteboul, S. Cluet, T. Milo. Correspondence and translation for heterogeneous data. *TCS* 275 (2002), 179–213.

2 A. Aho. Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science*, Volume A: 255-300, 1990.

3 R. Angles, C. Gutiérrez. Survey of graph database models. *ACM Comp. Surv.* 40(1):(2008).

4 P. Barceló, C. Hurtado, L. Libkin, P. Wood. Expressive languages for path queries over graph-structured data. In *PODS'10*, pages 3-14.

5 P. Barceló, L. Libkin, J. Reutter. Querying graph patterns. In *PODS'11*, pages 199–210.

6 J. Bhadra, A. Martin, J. Abraham. A formal framework for verification of embedded custom memories of the Motorola MPC7450 microprocessor. *Formal Methods in System Design* 27(1-2): 67-112 (2005).

7 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'00*, pages 176–185.

8 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443–465, 2002.

9 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Simplifying schema mappings. In *ICDT 2011*.

10 M. P. Consens, A. O. Mendelzon. Low complexity aggregation in GraphLog and Datalog. *TCS* 116 (1993), 95–116.

11 D. Freydenberger. Extended regular expressions: succinctness and decidability. in *STACS 2011*, pages 507-518.

12 W. Gelade, F. Neven. Succinctness of the complement and intersection of regular expressions. In *STACS 2008*, pages 325–336.

13 I. Glaister, J. Shallit. A lower bound technique for the size of nondeterministic finite automata. *IPL* 59:75-77, 1996.

14 O. Grumberg, O. Kupferman, S. Sheinvald. Variable automata over infinite alphabets. In *LATA'10*, pages 561–572.

15 C. Hagenah, A. Muscholl. Computing epsilon-free NFA from regular expressions in $O(n \log^2(n))$ time. In *MFCS'98*, pages 277–285.

16 T. Imielinski, W. Lipski. Incomplete information in relational databases. *J. ACM* 31 (1984), 761–791.

17 L. Kari, A. Mateescu, G. Paun, A. Salomaa. Multi-pattern languages. *TCS* 141 (1995), 253-268.

18 M. Kaminsky, D. Zeitlin. Finite-memory automata with non-deterministic reassignment. *IJFCS* 21 (2010), 741-760.

19 D. Kozen. Lower bounds for natural proof systems. In *FOCS'77*, pages 254-266.

20 Y. Liu, T. Rothamel, F. Yu, S. Stoller, N. Hu. Parametric regular path queries. In *PLDI'04*, pages 219–230.

21 Y. Liu, S. Stoller. Querying complex graphs. In *PADL'06*, pages 199–214.

22 A. R. Meyer, L. J. Stockmeyer. Word problems requiring exponential time. In *STOC 1973*, pages 1–9.

23 O. de Moor, D. Lacey, E. Van Wyk. Universal regular path queries. *Higher-Order and Symbolic Computation* 16(1-2): 15-35 (2003).

24 G. Pesant. A regular language membership constraint for finite sequences of variables. In *CP'04*, pages 482–295.

25 X. Shen, Y. Zhong, C. Ding. Predicting locality phases for dynamic memory optimization. *J. Parallel Distrib. Comput.* 67(7): 783-796 (2007).

Definable Operations On Weakly Recognizable Sets of Trees

Jacques Duparc[1], Alessandro Facchini[2], and Filip Murlak[3]

1 **University of Lausanne, Switzerland**
`jacques.duparc@unil.ch`
2 **University of Warsaw, Poland**
`facchini@mimuw.edu.pl`
3 **University of Warsaw, Poland**
`fmurlak@mimuw.edu.pl`

Abstract

Alternating automata on infinite trees induce operations on languages which do not preserve natural equivalence relations, like having the same Mostowski–Rabin index, the same Borel rank, or being continuously reducible to each other (Wadge equivalence). In order to prevent this, alternation needs to be restricted to the choice of direction in the tree. For weak alternating automata with restricted alternation a small set of computable operations generates all definable operations, which implies that the Wadge degree of a given automaton is computable. The weak index and the Borel rank coincide, and are computable. An equivalent automaton of minimal index can be computed in polynomial time (if the productive states of the automaton are given).

1998 ACM Subject Classification F.4.3 Formal languages, F.4.1 Mathematical Logic

Keywords and phrases alternating automata, Wadge hierarchy, index hierarchy

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.363

1 Introduction

The structure of a regular language of infinite trees can be analyzed in terms of recognizing automata, defining formulas, or topological properties. Each approach defines a hierarchy of classes of similar languages: the Mostowski–Rabin index hierarchy, the μ-calculus alternation hierarchy, the Borel hierarchy, the Wadge hierarchy. Sometimes complementary, sometimes closely related, together they approximate the missing canonical representation of regular languages. Understanding them has been a goal pursued for decades, bringing spectacular successes like the Wagner hierarchy for regular languages of infinite words [24], providing a full characterization of the topological and combinatorial structure of a language in terms of certain patterns in the recognizing deterministic automaton. Various versions of this pattern method were successfully applied to deterministic automata on trees, resulting in a full classification in terms of Wadge equivalence [16, 18], non-deterministic index [19], and weak alternating index [17].

Owing to the elegant correspondence between certain set-theoretical and ordinal operations [4], the whole Wadge hierarchy of Borel sets of finite rank can be generated with several simple operations, starting from the empty set. The pattern method builds on this result. In order to obtain lower bounds for the Wadge hierarchy of the considered class of automata, it is often enough to check that some operations are definable within the class [5, 6].

In obtaining upper bounds and computability results, the pattern method relies on certain compositionality of deterministic automata with respect to the equivalence relations of having

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 363–374

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the same Mostowski–Rabin index, the same Borel rank, or being continuously reducible to each other (Wadge equivalence). In a deterministic automaton each sub-automaton can be replaced with any automaton recognizing an equivalent language without influencing the equivalence class of the whole language. More generally, each automaton can be seen as a result of an operation performed on sub-automata by means of some connecting automaton. If the connecting automaton is deterministic, the operation induces an operation on the equivalence classes of the corresponding languages (see [16, 18], and also [19]). Sometimes, these operations can be expressed in terms of computable ordinal operations on Wadge degrees, and the degree of the recognized language can be obtained by bottom up evaluation starting from the simple sub-automata [16].

For alternating automata this approach fails in general, because the set-theoretical operation of union, easily simulated within an alternating automaton, is not an operation on the equivalence classes. Indeed, the union of arbitrarily complicated languages can be the whole space. Does it mean that the pattern method is confined to deterministic automata? Recently it has been shown that the method can be extended beyond deterministic automata, but the class of considered languages was very small [7]. In this paper we introduce a large syntactic class of the automata inducing operations compatible with the Wadge equivalence—we call them *game automata*—and show that it is the largest such class satisfying natural closure conditions (Sect. 4). We then focus on weak automata, and identify a small set of operations on Wadge equivalence classes which generate all other definable operations (Sect. 5). Based on this we show how to compute the Wadge degree and the Borel rank of weak game automata (Sect. 6). Finally, we prove that the Borel rank and the weak index coincide for weak game automata, which leads to an algorithm computing the weak index, and constructing the equivalent automaton with minimal index (Sect. 7).

Due to space limitations many proofs are moved to the full version of the paper [8].

2 Alternating Tree Automata

Let T_Σ denote the set of *(full infinite binary) trees* over an alphabet Σ, i.e., functions $t : \{0,1\}^* \to \Sigma$. Given $v \in \mathsf{dom}\,(t)$, by $t.v$ we denote the subtree of t rooted in v.

A *alternating tree automaton* $\langle \Sigma, Q, q_I, \delta, \mathsf{rank} \rangle$ consisting of a finite alphabet Σ, a finite set of states Q, an initial state $q_I \in Q$, a transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(\{0,1\} \times Q)$, where $\mathcal{B}^+(\{0,1\} \times Q)$ denotes the set of positive boolean formulae over $\{0,1\} \times Q$, and a rank function $\mathsf{rank} : Q \to \mathbb{N}$. As usual, A accepts $t \in T_\Sigma$ iff the player $\Diamond$ has a winning strategy in the induced max-parity game (see [8] for details). To underline this connection, we write transitions with $\Box$ and $\Diamond$ instead of $\wedge$ and $\vee$, e.g., $\delta(p,\sigma) = ((0,p) \Box (0,q)) \Diamond (1,q)$, or $p \xrightarrow{\sigma} ((0,p) \Box (0,q)) \Diamond (1,q)$. The class of all alternating automata is denoted by ATA.

An alternating tree automaton A is

- *weak* (WATA), if for all $q, q' \in Q$, if q is reachable from q' and q' is reachable from q, then $\mathsf{rank}\,(q) = \mathsf{rank}\,(q')$;
- *linear* (LATA), if for all $q, q' \in Q$, if q is reachable from q' and q' is reachable from q, then $q = q'$, and for each $q \in Q$ either all $\delta(q,\sigma)$ use only $\Box$ or all use only $\Diamond$;
- *deterministic* (DTA), if for all $q \in Q$, $\sigma \in \Sigma$, $\delta(q,\sigma) \in \big\{(0,p) \Box (1,q) \;\big|\; p,q \in Q\big\}$.

A state q is *reachable* from p if there exists a *path* in A from p to q, i.e., a sequence of states and alphabet symbols $p_0\sigma_0p_1\sigma_1 \dots p_{k-1}\sigma_k p_k$ such that $p_0 = p$, $p_k = q$, and p_{i+1} occurs in $\delta(p_i,\sigma_i)$ for all $i < k$. Throughout the paper we assume that all states are reachable from the initial state. By convention, $\top$ is a singled out all-accepting state, and $\bot$ is an all-rejecting state. We assume that all other states are *non-trivial*, i.e., accept some tree and

reject some tree. For every state q which is not the initial state q_I of the automaton A, by A_q we denote the automaton corresponding exactly to A except the fact that the initial state now is q and not q_I. We say that a state q of A is *productive* if $L(A_q) \neq \emptyset$.

The *(Mostowski–Rabin) index of an automaton* is given by $(i, j) \in \{0, 1\} \times \omega$, where i is the minimal and j is the maximal value of rank (scaling down the priorities we can always assume that the smallest rank is 0 or 1). Classes of languages recognizable with automata of index (i, j) form the so-called *index hierarchy*. By a result of Bradfield [3], we know that the index hierarchy for alternating tree automata, is strict. It is well-known that the class of weakly recognizable languages forms a strict hierarchy with respect to the index of the recognizing weak automata (cf. [1]). In the latter case we speak of the *weak index hierarchy*.

3 Borel classes and Wadge reductions

Consider the space T_Σ equipped with the standard Cantor (prefix) topology, that is the topology where a basic open set is the set all trees that extend a certain finite tree. Recall that the class of Borel sets of a topological space X is the closure of the class of open sets of X by countable unions and complementation. For a topological space X, the initial finite levels of the Borel hierarchy are defined as follows:

- $\mathbf{\Sigma}^0_1(X)$ is the class of open subsets of X,
- $\mathbf{\Pi}^0_n(X)$ contains complements of sets from $\mathbf{\Sigma}^0_n(X)$,
- $\mathbf{\Sigma}^0_{n+1}(X)$ contains countable unions of sets from $\mathbf{\Pi}^0_n(X)$.

By convention $\mathbf{\Sigma}^0_0(X) = \{\emptyset\}$ and $\mathbf{\Pi}^0_0(X) = \{X\}$.

The classes defined above are closed under inverse images of continuous functions. Let $\mathcal{C}$ be one of those classes. A set U is called $\mathcal{C}$-hard, if each set in $\mathcal{C}$ is an inverse image of U under some continuous function. If additionally $U \in \mathcal{C}$, U is said to be $\mathcal{C}$-complete. It is well known that every weakly recognizable tree language is a member of a Borel class of finite rank ([6, 14]). The rank of a language is the rank of the minimal Borel class the language belongs to. It can be seen as a coarse measure of complexity of languages.

A much finer measure of the topological complexity is the *Wadge degree*. If $T, U \subseteq T_\Sigma$, we say that T is *continuously (or Wadge) reducible* to U, $T \leqslant_W U$ in symbols, if there exists a continuous function f such that $T = f^{-1}(U)$. For a Borel class $\mathcal{C}$, T is $\mathcal{C}$-hard if $U \leqslant_W T$ for every $U \in \mathcal{C}$. We write $T \equiv_W U$ whenever $T \leqslant_W U \leqslant_W T$, and $T <_W U$, if $T \leqslant_W U$ but not $U \leqslant_W T$. The *Wadge hierarchy* is the partial order induced by $<_W$ on the $\equiv_W$-equivalence classes of Borel sets.

An alternative characterization of continuous reducibility can be given in terms of games. Let T and U be two arbitrary sets of trees. The *Wadge game* $\mathcal{W}(T, U)$ is played by two players, player I and player II. Each player builds a tree, say t_{I} and t_{II}, level by level. In every round, player I plays first, and both players add one level to their trees. Player II is allowed to skip her turn, but not forever. Player II wins the game if $t_{\mathrm{I}} \in T \Leftrightarrow t_{\mathrm{II}} \in U$.

▸ **Lemma 1** ([23]). *Let $T, U \subseteq T_\Sigma$. Then $T \leqslant_W U$ iff Player II has a winning strategy in the game $\mathcal{W}(T, U)$.*

An ordinal number is the order type of a well-ordered set. The least infinite ordinal is denoted by ω and corresponds to the order-type of the set of all natural numbers. We say that an ordinal α is countable if there is a bijection between α and ω. The first uncountable ordinal is denoted by ω_1. A subset B of an ordinal α is said to be *cofinal* if for every $a \in \alpha$ there exists some $b \in B$ such that $a \in b$. The *cofinality* of an ordinal α is thence the smallest ordinal β that is the order type of a cofinal subset of α.

Recall that a language L is called *self dual* if it is equivalent to its complement, otherwise it is called *non self dual*. From Borel determinacy [13], if $T, U \subseteq T_\Sigma$ are Borel, then $\mathcal{W}(T, U)$ is determined. As a consequence, a variant of Martin-Monk's result (cf. [11]) shows that $<_W$ is well-founded. Thus, we can associate to every Borel language an ordinal, called the *Wadge degree*, i.e. for sets of finite Borel rank, their Wadge degree is inductively defined by:

- $d_W(\varnothing) = d_W(\varnothing^{\complement}) = 1$,
- $d_W(L) = \sup\{d_W(K) + 1 \colon K \text{ non self dual}, K <_W L\}$ for $L >_W \varnothing$, non self-dual,
- $d_W(L) = \sup\{d_W(K) \colon K \text{ non self dual}, K <_W L\}$ for L self-dual.

For instance, open, non-closed sets have degree 2, just like closed, non-open sets. All clopens have degree 1. Let $\exp(\alpha) = \omega_1^\alpha$, and let ${}^{\omega_1}\epsilon_0 = \sup_{n\in\omega} \exp^n(\omega_1)$, the least fixpoint of the ordinal exponentiation of base ω_1. This is known to be the height of the Wadge hierarchy of all tree languages (recognizable or not) of finite Borel rank. More precisely, if L is $\mathbf{\Sigma}^0_n$-complete for $n > 1$, then $d_W(L) = \exp^{n-1}(1)$ for $n > 1$ (cf. [4]).

For each degree there are exactly three equivalence classes with the same degree, represented by U, $U^{\complement}$ and $U^{\pm} = \{t \mid t(\epsilon) = a, t.0 \in U\} \cup \{t \mid t(\epsilon) \neq a, t.0 \notin U\}$ for some non self-dual set U and $a \in \Sigma$. It easy to check that $U, U^{\complement} <_W U^{\pm}$ and $U^{\pm}$ is self-dual.

For each non self-dual set one can determine its sign, $+$ or $-$, which specifies precisely the $\equiv_W$-class [4]. For sets $U \subseteq T_\Sigma$ with $d_W(U)$ of countable cofinality, the sign is $+$ if U is Wadge equivalent to the set of trees over $\Sigma \cup \{c\}$, $c \notin \Sigma$, which have no c on the leftmost branch, or the first c is in the node 0^i and $t.0^i \in U$. The sign is $-$ if U is equivalent to the complement of this set. For instance, $\varnothing$ and open, non-closed sets have sign -, while the whole space and closed, non-open sets have sign $+$. For sets of cofinality ω_1, the definition is more complicated, but $\mathbf{\Sigma}^0_n$-complete sets have sign $-$, and $\mathbf{\Pi}^0_n$-complete sets have sign $+$. All self-dual sets by definition have sign $\pm$. Thus an ordinal $\alpha < {}^{\omega_1}\epsilon_0$ and a sign $\epsilon \in \{+, -, \pm\}$, determine a $\equiv_W$-class, denoted $[\alpha]^\epsilon$.

4 Game automata

For A, B (over the same alphabet) and an occurrence of a state q in a transition $\delta(p, \sigma)$ of A, the substitution A_B is obtained by replacing the occurrence of q in $\delta(p, \sigma)$ with the initial state of B. The mapping $B \mapsto A_B$ induces an operation on recognized languages, but it need not preserve coarser equivalence relations, like Wadge equivalence.

As pointed out in the introduction, the operation of union is not compatible with such equivalence relations. The same is true of intersection.

▸ **Example 2.** Take $\Sigma = \{0, 1, 2\}$ and consider $(\Sigma^*(1 + 2))^\omega$ and $(\Sigma^*2)^\omega$. Clearly, $(\Sigma^*(1 + 2))^\omega \leqslant_W (\Sigma^*2)^\omega$ as witnessed by the letter-to-letter morphism $0 \mapsto 0$ and $1, 2 \mapsto 2$. The converse reduction is given by the inclusion. Taking union with Σ^*0^ω, we obtain $(\Sigma^*(1 + 2))^\omega \cup \Sigma^*0^\omega = \Sigma^\omega$, and $(\Sigma^*2)^\omega \cup \Sigma^*0^\omega \not\equiv_W \Sigma^\omega$. The language $(\Sigma^*2)^\omega \cup \Sigma^*0^\omega$ is at the level $\mathbf{\Delta}^0_3$ of the Borel hierarchy, a deterministic automaton requires three ranks to recognize it, and an alternating automaton needs two. This makes it much more complex than the whole space Σ^ω, which can be recognized by a deterministic automaton with a single state, whose rank is 0. Similarly, intersecting with $\Sigma^*(0 + 1)^\omega$ we obtain $\Sigma^*(0^*1)^\omega$, and the empty set, which have very different complexity.

In order to ensure that substitution is well-behaved, we need to prevent the automata from simulating union and intersection. We call a transition $\delta(q, a)$ *ambiguous* if it contains two occurrences of some direction $d \in \{0, 1\}$.

▸ **Fact 3.** Let $\mathcal{C} \subseteq \mathsf{ATA}$ be a class of automata over a fixed alphabet with at least two letters, closed under substitution and containing the one-state all-rejecting and all-accepting automata. Substitution preserves the Wadge equivalence in $\mathcal{C}$ iff no automaton of $\mathcal{C}$ has an ambiguous transition.

Proof. Assume for simplicity that the alphabet contains the symbols $0, 1, 2$. Starting from the all-accepting and all-rejecting automata over the alphabet $\{0, 1, 2\}$ we can obtain automata $A, A^{\complement}B, B^{\complement}$ recognizing languages L_{0^ω}, $(L_{0^\omega})^{\complement}$, $L_{(0+1)^\omega}$, $(L_{(0+1)^\omega})^{\complement}$ respectively, where L_α stands for the set of trees whose leftmost branch is a word from the language defined by the expression α. Observe that $L(A) \equiv_W L(B)$, but $L(A) \cup L(B^{\complement}) \not\equiv_w L(B) \cup L(B^{\complement})$ and $L(A) \cap L(A^{\complement}) \not\equiv_w L(B) \cap L(A^{\complement})$.

Let $C \in \mathcal{C}$ and let $q_0\sigma_0 q_1\sigma_1 \dots q_k$ be path from the initial state q_0 to a state q_k such that for some σ_k, $\delta(q_k, \sigma_k)$ is an ambiguous transition. By substituting the all-accepting and all-rejecting automata, we can assume that $\delta(q_i, \sigma_i) = (d_i, q_{i+1})$ for $i < k$ and $\delta(q_k, \sigma_k) = (d_k, p_0) \diamond (d_k, p_1)$ or $\delta(q_k, \sigma_k) = (d_k, p_0) \square (d_k, p_1)$ for some states p_0, p_1. Assume that $\delta(q_k, \sigma_k) = (d_k, p_0) \diamond (d_k, p_1)$, and let C' be the result of replacing the occurrence of p_0 with the initial state of B, and the occurrence of p_1 with the initial state of $B^{\complement}$. For C'_A, obtained by replacing the initial state of B with the initial state of A, we have $L(C'_A) \equiv_W L(A) \cup L(B^{\complement})$, and $L(C') \equiv_W L(B) \cup L(B^{\complement})$, which concludes the proof. For $\delta(q_k, \sigma_k) = (d_k, p_0) \square (d_k, p_1)$, use $A^{\complement}$ instead of $B^{\complement}$. ◂

Observe that each non-ambiguous transition has one of the four forms: $(0, p)$, $(1, p)$, $(0, p) \diamond (1, q)$, or $(0, p) \square (1, q)$.

▸ **Definition 4.** A *game automaton* (GA) is an alternating automaton without ambiguous transitions. For notational simplicity, we assume that

$$\delta \colon Q \times \Sigma \to \{p \diamond q \mid p, q \in Q \backslash \{\top\}\} \cup \{p \square q \mid p, q \in Q \backslash \{\bot\}\} ,$$

where $p \diamond q$ and $p \square q$ is interpreted as $(0, p) \diamond (1, q)$ and $(0, p) \square (1, q)$, respectively.

A *weak game automaton* (WGA), is a game automaton which is also weak, and a *linear game automaton* (LGA) [7], is a game automaton which is linear.

Fact 3 implies that GA it the largest nontrivial subclass of ATA closed under substitution for which substitution preserves Wadge equivalence, and similarly for WGA $\subseteq$ WATA. In fact, a more general property holds for GA.

▸ **Fact 5.** For every GA A, B, B', every state q of A, and A_B, $A_{B'}$ obtained by replacing an occurrence of q with the initial state of B and B' respectively, it holds that

1. $L(B) \leqslant_W L(B')$ implies $L(A_B) \leqslant_W L(A_{B'})$,
2. $L(A_q) \leqslant_W L(A)$.

Relations between the classes are shown in Fig. 1 with arrows standing for class inclusion. The classes GA, WGA, and LGA are closed under complementation: the usual complementation procedure of increasing the ranks by one and swapping existential and universal transitions works. However they are neither closed under union nor intersection. For instance, let $L_\sigma = \{t \in T_{\{a,b\}} \colon t(0) = t(1) = \sigma\}$. Obviously, L_a and L_b are LGA-recognizable, but $L_a \cup L_b$ is not even GA-recognizable. Note that the last example also shows that all the inclusions in the diagram above are strict.

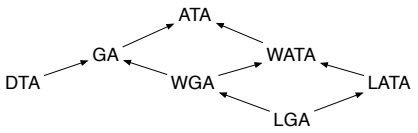


Figure 1

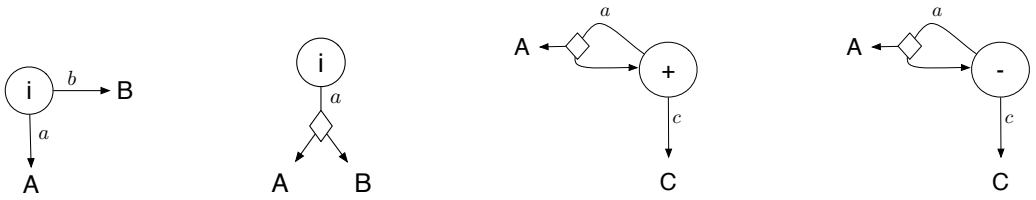


Figure 2 Automata constructions for $\sqcup$, $\diamond$, loop^+, $\exists$.

5 Operations induced by automata

LGA, investigated in [7], can be classified in terms of several simple set theoretic operations (we assume that the alphabet contains letters a, b, c):

$$\begin{aligned}
L \sqcup M &= \{t \mid t(\varepsilon) = a, t.0 \in L\} \cup \{t \mid t(\varepsilon) \neq a, t.0 \in M\}\ , \\
L \square M &= \{t \mid t.0 \in L \wedge t.1 \in M\}\ , \\
L \diamond M &= \{t \mid t.0 \in L \vee t.1 \in M\}\ , \\
\mathsf{loop}^-(L,M) &= \bigcup_{n\in\mathbb{N}} \{t \mid \text{first } c \text{ is in } 0^n, t.0^{n+1} \in M, \text{ and } t.0^\ell 1 \in L \text{ for all } \ell < n\}\ , \\
\mathsf{loop}^+(L,M) &= \bigcup_{n\in\mathbb{N}} \{t \mid \text{first } c \text{ is in } 0^n, \text{ and } t.0^{n+1} \in M \text{ or } t.0^\ell 1 \in L \text{ for some } \ell < n\} \cup \\
&\quad \cup \{t \mid t.(0^n) \neq c \text{ for all } n\}\,, \\
\forall(L,M) &= \mathsf{loop}^-(L,M) \cup \{t \mid t.(0^n) \neq c \text{ for all } n, \text{ and } t.0^\ell 1 \in L \text{ for all } \ell\}\,, \\
\exists(L,M) &= \mathsf{loop}^+(L,M) \cup \{t \mid t.(0^n) \neq c \text{ for all } n, \text{ and } t.0^\ell 1 \in L \text{ for some } \ell\}\,,
\end{aligned}$$

where "first c is in 0^n" means that $t(0^n) = c$ and $t(0^k) \neq c$ for all $k < n$. Observe that $(L \square M)^{\complement} = L^{\complement} \diamond M^{\complement}$, $(\mathsf{loop}^+(L,M))^{\complement} = \mathsf{loop}^-(L^{\complement}, M^{\complement})$, and $(\forall(L,M))^{\complement} = \exists(L^{\complement}, M^{\complement})$.

These operations are definable by LGA: automata realizations for $\sqcup$, $\diamond$, loop^+, $\exists$ are shown in Fig. 2, and for $\square$, loop^-, $\forall$ they are obtained by replacing $\diamond$ with $\square$ and swapping the rank parities. Like all operations induced by GA, they are compatible with Wadge equivalence.

▸ **Fact 6.** Let op be one of the operations $\sqcup$, $\diamond$, loop^+, $\exists$, or their duals. Whenever $L \equiv_W L'$ and $M \equiv_W M'$, it holds that $\mathsf{op}(L,M) \equiv_W \mathsf{op}(L',M')$.

Up to Wadge equivalence, these operations are everything LGA are able to express.

▸ **Fact 7** ([7])**.** Up to Wadge equivalence, the closure of $\{\top, \bot\}$ under $\sqcup$, $\diamond$, loop^+, $\exists$, and their duals (or equivalently, complementation) gives exactly the family of sets recognized by LGAs. Moreover, for each LGA one can compute an equivalent canonical term over these operations and $\bot$, $\top$.

Since the operations preserve Wadge equivalence, they can be defined in terms of ordinal arithmetics and signs [4, 7]. For some operations the definitions are very simple, for instance

$$[\gamma_1]^{\epsilon_1} \sqcup [\gamma_2]^{\epsilon_2} = [\max(\gamma_1,\gamma_2)]^{\epsilon}, \text{ where } \epsilon = \begin{cases} \epsilon_1 & \text{if } \gamma_1 > \gamma_2 \\ \pm & \text{if } \gamma_1 = \gamma_2 \text{ and } \epsilon_1 \neq \epsilon_2, \\ \epsilon_2 & \text{otherwise} \end{cases}$$

$$\mathsf{loop}^+([\gamma]^\epsilon, [1]^-) = \left[\sup_k d_W(([\gamma]^\epsilon)^{\langle k\rangle})\right]^+, \text{ where } U^{\langle k\rangle} = \underbrace{U \diamond U \diamond \cdots \diamond U}_{k}$$

$$\exists([\gamma]^\epsilon, [1]^-) = [\exp^{i+1} 1]^-, \text{ for } [\exp^i 1]^+ \leqslant_W [\gamma]^\epsilon \leqslant_W [\exp^{i+1} 1]^-\,.$$

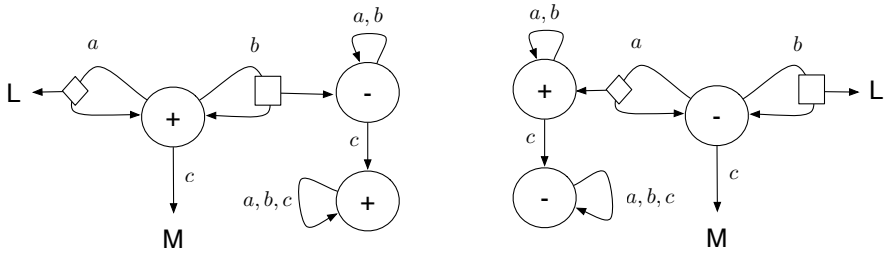


Figure 3 Operations definable with WGA.

Observe that the second equation, and its dual, imply that for all k

$$\mathsf{loop}^+(L,M) \geqslant_W L^{\langle k\rangle}, \qquad \mathsf{loop}^-(L,M) \geqslant_W L^{[k]},$$

where $U^{[k]} = \underbrace{U \,\square\, U \,\square \cdots \square\, U}_{k}$. For $\diamond$ the ordinal definition has only been given for $\equiv_W$-classes inhabited by LGA-recognizable languages, $[\Phi] = \{[\alpha]^\epsilon \mid \alpha \in \Phi, \epsilon \in \{+,-,\pm\}\}$ with Φ denoting the set of ordinals of the form $\sum_{n=N}^{0} \beta_n + \alpha$ where $\alpha < \omega$ and each β_n is of the form $\exp^n(\omega)\eta + \sum_{p=P}^{1} \exp^n(p)k_p$ for some $\eta < \omega^\omega$ and $k_p < \omega$. Closure of $[\Phi]$ under $\sqcup$, $\diamond$, loop^+, $\exists$ (and their duals) was the technical core of the proof of Fact 7.

In this work we want to move to sets recognizable by WGA. Surprisingly, only two really new operations are introduced, $\mathsf{loop\text{-}reset}^+(L,M)$ and $\mathsf{loop\text{-}reset}^-(L,M)$. The automata constructions for them are shown in Fig. 3.

By a Wadge game argument we get a simple characterization in terms of ordinal arithmetics, showing that WGA-definable operations can multiply some Wadge degrees by ω_1.

▸ **Theorem 8.** *For every Wadge equivalence class $[\gamma]^\epsilon$ of a Borel language and $\mu \in \{+,-\}$*

$$\mathsf{loop\text{-}reset}^\mu([\gamma]^\epsilon, [1]^{\overline{\mu}}) = \begin{cases} [3]^{\overline{\mu}} & \text{if } [\gamma]^\epsilon \equiv_W [1]^\mu, \\ \left[d_W(\mathsf{loop}^+([\gamma]^\epsilon, [1]^-))\omega_1\right]^\mu & \text{otherwise}, \end{cases}$$

where $\overline{\mu} = +$ if $\mu = -$, $\overline{\mu} = -$ otherwise.

This operation is the source of difference between LGA and WGA, and allows WGA to inhabit much many Wadge equivalence classes than LGA. Thus, in our algorithm for WGA we use effective closure for a larger set of ordinals. Let Ω be the set of ordinals of the form $\Sigma_{i=K}^{0} \exp(\alpha_i)\eta_i$ where $\alpha_K, \alpha_{K-1}, \ldots, \alpha_0$ is a strictly decreasing sequence of ordinals from Φ, and $\eta_i < \omega$ for $\mathrm{cof}\alpha_i = \omega_1$ or $\mathrm{cof}\alpha_i < \omega$, and $\eta_i < \omega^\omega$ for $\mathrm{cof}\alpha_i = \omega$.

▸ **Lemma 9.** *$[\Omega]$ is closed under the operations $\sqcup$, loop^+, $\mathsf{loop\text{-}reset}^+$, $\exists$ (and their duals) and the result of the operation can be computed effectively.*

The proof is by induction, with the base cases covered by the closure property for $[\Phi]$.

6 Computing the Wadge degrees of WGA

For game automata, a run (computation tree) over an input tree t is a labeling of the input tree with states and modes ($\square$ or $\diamond$), induced by the transition function of the automaton. A transition taken from a node v determines the mode of v and the states in its children as follows: the root is labelled with the initial state, and if a node with label σ gets state q and

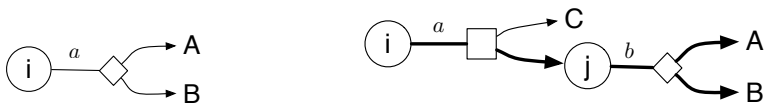


Figure 4 A simulation (the rank j must not be greater then i).

$q \xrightarrow{\sigma} q' \circ q''$ then v gets the mode $\circ$, and the left and right children get the states q' and q'' respectively. A run ρ is *resolved up to* a subtree ρ' if for all $v, v0, v1 \in \mathsf{dom}\,\rho$ such that exactly one node vd belongs to $\mathsf{dom}\,\rho'$, and for the remaining node vd' the sub-run $\rho.vd'$ is accepting if v's mode is $\Box$ and rejecting if it is $\Diamond$.

▸ **Definition 10.** A *simulation* of a run ρ in a run σ is a partial function $\eta : \mathsf{dom}\,\rho \to \mathsf{dom}\,\sigma$ such that

- $\mathsf{dom}\,\eta$ is a prefix closed subset of $\mathsf{dom}\,\rho$ (possibly with leaves and infinite branches);
- σ is resolved up to the subtree induced by the image of η;
- for each $v0, v1 \in \mathsf{dom}\,\eta$, $\eta(v0), \eta(v1)$ are descendants of $\eta(v)$, their closest common ancestor has the same mode as v , and the highest rank on the path from $\eta(v)$ to $\eta(vd)$ is equal to the rank of state in vd for $d = 0, 1$;
- for each leaf $v \in \mathsf{dom}\,\eta$, $\rho.v$ is accepting iff $\sigma.\eta(v)$ is accepting.

▸ **Lemma 11.** *If there is a game simulation of ρ in σ, then ρ is accepting iff σ is accepting.*

Proof. Each strategy in the parity game on ρ can be carried over to σ, and *vice versa*. ◂

▸ **Definition 12.** A *simulation* of an automaton A in an automaton B consists of a partition of Q^A into sets Q_1, Q_2, Q_3 and function $\eta\colon Q_1 \cup Q_2 \to Q^B$ such that

- $q_I^A \in Q_1$ and each transition of A originating in Q_1 leads to $Q_1 \cup Q_2$;
- whenever $q \xrightarrow{\sigma}_A q_0 \circ q_1$ for some $q \in Q_1$ and $\circ \in \{\Diamond, \Box\}$, there exist a path π from $\eta(q)$ to some p and paths π_i from some p_i to $\eta(p_i)$ for $i = 0, 1$ such that $p \xrightarrow{\tau}_B p_0 \circ p_1$ or $p \xrightarrow{\tau}_B p_1 \circ p_0$ and the highest rank on $\pi\tau\pi_i$ is equal to $\mathsf{rank}\,q_i$;
- for all $q \in Q_2$, $L(A_q) \leqslant_W L(B_{\eta(q)})$.

An example of a simulation is given in Fig. 4. A simulation of A in B immediately provides a continuous reduction from the set of accepting runs of A to the set of accepting runs of B. The next lemma follows by noticing that for GAs the set of accepting runs is Wadge equivalent to the recognized language.

▸ **Lemma 13.** *If there exists a simulation of A in B, then $L(A) \leqslant_W L(B)$.*

Strongly connected components (SCCs) of automata are defined as for graphs in terms of reachability. An SCC is *trivial* if it does not contain any loop. A transition $q \xrightarrow{\sigma} q' \circ q''$ is called *branching* if q, q', q'' belong to the same SCC.

▸ **Lemma 14.** *For each* WGA *one can effectively compute a Wadge equivalent* WGA *over $\{a, b, c\}$ without non-trivial loops.*

Proof. First we construct an automaton over a larger alphabet. We collapse each strongly connected component into one state, proceeding by induction on the DAG of SCCs. Let X be the root SCC, i.e., the SCC containing the initial state q_I. By induction hypothesis, we can assume that all other SCCs consist of a single state.

If there is a branching $\Box$-transition in X, set $q_I \xrightarrow{a} q_I \,\Box\, q_I$. Otherwise, set $q_I \xrightarrow{a_p} q_I \,\Box\, p$ for all $p \notin X$ such that $q \xrightarrow{\sigma} q' \,\Box\, p$ or $q \xrightarrow{\sigma} p \,\Box\, q'$ for some $q, q' \in X$. Define the transitions via b and b_p analogously, replacing $\Box$ with $\Diamond$. Finally, let $q_I \xrightarrow{c_{p \circ p'}} p \circ p'$ where $p \circ p'$ ranges over

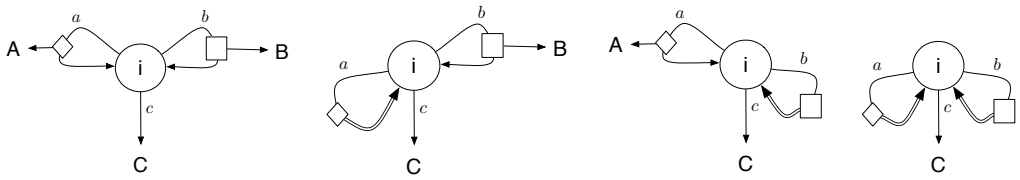


Figure 5 Strongly connected components of WGA over $\{a, b, c\}$ without non-trivial loops.

- $p \circ p'$ such that $p, p' \notin X$ and $q \xrightarrow{\sigma} p \circ p'$ for some $q \in X$, $\circ \in \{\diamond, \square\}$;
- $p \square \top$ such that $p \notin X$ and $q \xrightarrow{\sigma} q' \square p$ or $q \xrightarrow{\sigma} p \square q'$ for some $q, q' \in X$; and
- $p \diamond \bot$ such that $p \notin X$ and $q \xrightarrow{\sigma} q' \diamond p$ or $q \xrightarrow{\sigma} p \diamond q'$ for some $q, q' \in X$.

For each state q of the new automaton, extend $\delta(q, \sigma)$ to all symbols in the alphabet by using one of already defined transitions.

The original automaton can be simulated in the modified one by taking $Q_1 = X$, $Q_2 = \{p \mid q \xrightarrow{\sigma} p \circ r \text{ or } q \xrightarrow{\sigma} r \circ p \text{ for some } q \in X, r \in Q\}$, and $\eta(q) = q_I$ for $q \in Q_1$, and $\eta(p) = p$ for $p \in Q_2$. For the converse simulation, only change Q_1 to $\{q_I\}$, and for Q_2 and η keep the definitions above.

To reduce the alphabet to $\{a, b, c\}$, modify the construction as follows. In the case where there is no branching $\square$-transition in X, add a single transition $q_I \xrightarrow{a} q_I \square q_a$, where q_a is the initial state of the automaton recognizing $L(A_{p_1}) \sqcup L(A_{p_2}) \sqcup \cdots \sqcup L(A_{p_k})$, where $\{p_1, p_2, \ldots, p_k\}$ is the set over which p ranges in the original construction. For b the modification is analogous, and for c add $q_I \xrightarrow{\sigma} \bot \diamond q_c$, where q_c is the initial state of the automaton recognizing $\big[L(A_{p_1}) \circ_1 L(A_{p'_1})\big] \sqcup \big[L(A_{p_2}) \circ_2 L(A_{p'_2})\big] \sqcup \cdots \sqcup \big[L(A_{p_k}) \circ_\ell L(A_{p'_\ell})\big]$, where $p_1 \circ_1 p'_1, p_2 \circ_2 p'_2, \ldots, p_\ell \circ_\ell p'_\ell$ are the triples over which $p \circ p'$ ranges in the original construction. Observe that these modifications do not influence the Wadge equivalence class of the recognized language. ◂

Thus we can assume that each non-trivial SCC of a given WGA is of one of the four forms presented in Fig. 5. By a Wadge game argument we can further simplify the automaton.

▸ **Lemma 15.** *For each* WGA *one can compute effectively a Wadge equivalent* WGA *over* $\{a, b, c\}$ *without non-trivial loops and branching transitions (except those for* $\top$, $\bot$*).*

After these simplifications we apply Lemma 9 to compute the Wadge degrees.

▸ **Theorem 16.** *For a given* WGA *one can effectively compute the Wadge equivalence class of the language it recognizes.*

Proof. By Lemma 15, we can assume that the automaton is over $\{a, b, c\}$, has no non-trivial loops, and no branching transitions. By induction on the DAG of SCCs we prove that the Wadge equivalence class of the recognized language is in $[\Omega]$ and can be computed effectively.

If the whole automaton consists of a single SCC, the result is $[1]^+$ or $[1]^-$ depending on the rank of the unique state.

To perform the inductive step, it suffices to express the recognized language in terms of the operations from Lemma 9. If there is no transition from the initial state q_I that leads back to q_I, the recognized language can be presented as $\big[L(A_{p_a}) \circ_a L(A_{p'_a})\big] \sqcup \big[L(A_{p_b}) \circ_b L(A_{p'_b})\big] \sqcup \big[L(A_{p_c}) \circ_c L(A_{p'_c})\big]$, where $q_I \xrightarrow{\sigma} p_\sigma \circ_\sigma p'_\sigma$ for $\sigma = a, b, c$.

For the rest of the proof we assume that the automaton is of the form shown in the leftmost part of Fig. 5; we use the notation $q_I(A, B, C)$. For the remaining possibilities the computations are analogous. If the $\mathsf{rank}\, q_I$ is even, consider the following cases.

1. $L(A) \geqslant_W \forall(L(B), \top)$. Then $L(A) >_W L(B)^{[n]}$ for every $n < \omega$, and we have that either $L(q_I(A, B, C)) \equiv_W L(q_I(A, B', C))$ for some B' recognizing a $\mathbf{\Sigma}^0_1$-complete language, if $d_W(L(B)) \geqslant [2]^-$, or $L(q(A, B, C)) \equiv_W L(q_I(A, \top, C))$ otherwise. In the former case the recognized language is Wadge equivalent to $\mathsf{loop\text{-}reset}^+(L(A), L(C))$, in the latter case it is Wadge equivalent to $\mathsf{loop}^+(L(A), L(C))$.
2. $L(A) <_W L(\forall(B, \top))$. The recognized language is Wadge equivalent to $L(q_I(\bot, B, C))$, which gives $\forall(L(B), L(C))$.
3. $L(A) \equiv_W L(\forall(B, \top))^{\complement}$. In this case, as $L(A) >_W L(B)^{[n]}$ for every $n < \omega$, we conclude that the recognized language is Wadge equivalent to $L(A) \diamond L(q(\bot, B, C)) \equiv_W L(A) \diamond \forall(L(B), L(C))$.

For $\mathsf{rank}\, q_I$ odd, dualize the above argument. ◂

7 Borel rank and weak index

As an immediate corollary of Theorem 16 we obtain decidability of the Borel rank problem.

▸ **Corollary 17.** *The problem of deciding the Borel rank of a* WGA*-recognizable language is decidable.*

We will now proceed to prove that the weak index conjecture holds for languages recognized by WGA. It has long been known that one implication holds.

▸ **Proposition 18** ([14])**.** Let $A \in \mathsf{WGA}$ with index $(0, n)$ (resp. $(1, n+1)$). Then it holds that $L(A) \in \Pi^0_n$ (resp. $L(A) \in \Sigma^0_n$).

Using the connections between the structure and topological complexity of automata explained in the previous sections, we can prove the converse for WGA.

▸ **Theorem 19.** *For languages recognizable by* WGA*, the Borel hierarchy and the weak index hierarchy coincide.*

Proof. By duality and Proposition 18 it suffices to show that each WGA A recognizing a $\mathbf{\Pi}^0_n$ language admits an equivalent WATA of index $(0, n)$. We proceed by induction on the DAG of SCCs of the automaton.

If $n = 0$, A accepts every tree, so it is equivalent to a single state automaton of index $(0, 0)$. If $n = 1$, A cannot contain a productive state reachable from a nontrivial rejecting SCC, so an equivalent $(0, 1)$ automaton can be obtained by setting the rank of all states reachable from non-trivial rejecting SCCs to 1 and the rank of the remaining states to 0.

Suppose that $n \geqslant 2$, and let X be the root SCC. If X has rank 0 (we can change it to 0 if X is trivial), by Fact 5 (2) and the induction hypothesis we can present all A_q with $q \notin X$ as $(0, n)$ automata and the claim follows.

Suppose X is non-trivial and has rank 1. Assume that X contains a branching $\diamond$-transition. Then it follows that for all states q, $L(A_q)$ is in $\mathbf{\Sigma}^0_{n-1}$ (otherwise, the whole language would be $\mathbf{\Sigma}^0_n$ hard). In consequence, for all states $p \notin X$, A_p can be transformed into an equivalent WATA of index $(1, n)$, and we conclude like before.

The remaining case is that of non-trivial X of rank 1, without branching $\diamond$-transitions. Observe that in this case, there are two reasons why a tree can be rejecting:

1. a path of the computation stays forever in X, and for all $\diamond$ transitions in this path, the branches leaving X are rejecting;
2. a rejecting path exits X, and for all $\diamond$ transitions in this path, branches leaving X are rejecting.

By induction hypothesis, all A_p can be transformed to WATA of index $(1, n)$ if $q \xrightarrow{\sigma} p \diamond q'$ or $q \xrightarrow{\sigma} q' \diamond p$ for some $q, q' \in X$, or $(0, n)$ otherwise. To check that the second condition does not hold, use A with the rank X changed to 0. For the first condition, use A' obtained from A by replacing $q \xrightarrow{\sigma} p \circ p'$ with $q \xrightarrow{\sigma} \bot \diamond \bot$, $q \xrightarrow{\sigma} p \square q'$ with $q \xrightarrow{\sigma} \top \diamond q'$, and $q \xrightarrow{\sigma} q' \square p'$ with $q \xrightarrow{\sigma} q' \diamond \top$ for all $q, q' \in X$, $p, p' \notin X$. The ϵ-transition introduced to implement conjunction can be removed by unraveling the first step of the computation, without changing the ranks. ◂

This way the weak index problem reduces to the Borel rank problem. The construction above in fact gives an effective way of constructing the equivalent WATA of minimal index.

▸ **Corollary 20.** *The problem of calculating the exact position in the weak index hierarchy of a language recognized by a* WGA *is decidable and an equivalent* WATA *can be constructed effectively (in polynomial time if the productive states are given).*

8 Conclusions

We have isolated the class of game automata, a wide class of automata inducing operations on Wadge equivalence classes. For *weak* game automata we were able to use this property to describe all definable operations in terms of a small set of generators, and based on this we gave a procedure calculating the Wadge equivalence class of the language recognized by any given automaton. Using the structural information provided by the latter result we proved that the weak index hierarchy and the Borel hierarchy coincide, and gave algorithms computing the weak index and constructing an equivalent weak alternating automaton of the minimal index.

The results on the Wadge hierarchy subscribe to the line of reaserch aimed at investigating the hierarchies for families of languages recognized by various devices (cf. [5, 9, 21]). Usually, lower bounds on the heights of the hierarchies are easier to obtain, tight upper bounds are more difficult, and decidability results are scarce [7, 16, 24]. The peculiarity of this work is that we obtain computability of the Wadge degree without determining explicitly the inhabited levels of the hierarchy. Some lower bounds are easy to obtain based on our description of the induced operations and an upper bound is given by $[\Omega]$, but giving a full characterization of the inhabited levels seems to be a nontrivial task.

The class of automata we are considering has limited expressivity, but it seems to capture many interesting topological phenomena. Even more so in the unrestricted case, as game automata recognize the game languages recently considered by Arnold and Niwiński [2] in their study of the Wadge hierarchy of non-Borel regular languages. Currently, we are trying to drop the weakness restriction. One of the challenges is that for non-Borel languages the shape of Wadge hierarchy is unknown.

Despite the positive results concerning the hierarchy problems for weak game automata, and hopefully for non-weak, from the methodological point of view the message of this work is that we are reaching the limits of the topological approach to index problems. Pushing decidability results beyond game automata seems to require new techniques.

Acknowledgements

The second author is supported by a grant from the SNFS, n. PBLAP2-132006, while the third author is supported by the Polish government grant no. N N206 567840.

References

1 A. Arnold, J. Duparc, F. Murlak, D. Niwiński. On the Topological Complexity of Tree Languages. In J. Flum et al. (Eds.) *Logic and Automata - History and Perspectives*, Texts in Logic and Games, Amsterdam University Press: 9–28 (2007).

2 A. Arnold, D. Niwiński. Continuous Separation of Game Languages. *Fund. Info.*, 81(1–3): 19–28 (2008).

3 J. Bradfield. The Modal μ-Calculus Alternation Hierarchy is Strict. *Theor. Comput. Sci.* 195(2): 133–153 (1998).

4 J. Duparc. Wadge Hierarchy and Veblen Hierarchy Part 1: Borel Sets of Finite Rank. *J. Symb. Log.* 66(1): 56–86 (2001).

5 J. Duparc. A Hierarchy of Deterministic Context-Free ω-Languages. *Theoret. Comput. Sci.* 290:1253–1300 (2003).

6 J. Duparc, F. Murlak. On the Topological Complexity of Weakly Recognizable Tree Languages. *FCT*: 261–273 (2007).

7 J. Duparc, A. Facchini, F. Murlak. Linear Game Automata: Decidable Hierarchy Problems for Stripped-Down Alternating Tree Automata. *CSL*: 225–239 (2009).

8 J. Duparc, A. Facchini, F. Murlak. Definable Operations On Weakly Recognizable Sets of Trees. `http://www.mimuw.edu.pl/~fmurlak/papers/gamafull.pdf`.

9 O. Finkel. Borel Ranks and Wadge Degrees of ω-Context Free Languages. *Mathematical Structures in Computer Science* 16: 813–840 (2006).

10 S. Hummel, H. Michalewski, D. Niwiński. On the Borel Inseparability of Game Tree Languages. *STACS*: 565–576 (2009).

11 A. S. Kechris. *Classical Descriptive Set Theory.* Graduate Texts in Mathematics, Vol. 156. Springer-Verlag, New York (1995).

12 L. H. Landweber. Decision Problems for ω-Automata. *Math. Systems Theory* 3: 376–384 (1969).

13 D. A. Martin. Borel Determinacy. *Ann. of Math.* (2), 102(2): 363–371 (1975).

14 A. W. Mostowski. Hierarchies of Weak Automata and Weak Monadic Formulas. *Theoret. Comput. Sci.* 83: 323–335 (1991).

15 F. Murlak. On Deciding Topological Classes of Deterministic Tree Languages. *CSL*: 573–584 (2005).

16 F. Murlak. The Wadge Hierarchy of Deterministic Tree Languages. *Logical Methods in Comput. Sci.*, 4(4), Paper 15.

17 F. Murlak. Weak Index vs Borel Rank. *STACS*: 573–584 (2008).

18 D. Niwiński, I. Walukiewicz. A Gap Property of Deterministic Tree Languages. *Theor. Comput. Sci.* 303: 215–231 (2003).

19 D. Niwiński, I. Walukiewicz. Deciding Nondeterministic Hierarchy of Deterministic Tree Automata. *Electr. Notes Theor. Comput. Sci.* 123: 195–208 (2005).

20 M. O. Rabin. Decidability of second-order theories and automata on infinite trees. Trans. Amer. Soc. **141**: 1–35 (1969).

21 V. Selivanov. Wadge Degrees of ω-Languages of Deterministic Turing Machines. *Theoret. Informatics Appl.*, 37: 67–83 (2003).

22 J. Skurczyński. The Borel Hierarchy is Infinite in the Class of Regular Sets of Trees. *Theoret. Comput. Sci.* 112: 413–418 (1993).

23 W. W. Wadge. *Reducibility and Determinateness on the Baire Space.* Ph.D. Thesis, Berkeley (1984).

24 K. Wagner. On ω-Regular Sets. *Inform. and Control* 43: 123–177 (1979).

Nash Equilibria in Concurrent Games with Büchi Objectives

Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels

LSV, CNRS & ENS Cachan, France
{bouyer,brenguier,markey,ummels}@lsv.ens-cachan.fr

Abstract

We study the problem of computing pure-strategy Nash equilibria in multiplayer concurrent games with Büchi-definable objectives. First, when the objectives are Büchi conditions on the game, we prove that the existence problem can be solved in polynomial time. In a second part, we extend our technique to objectives defined by deterministic Büchi automata, and prove that the problem then becomes EXPTIME-complete. We prove PSPACE-completeness for the case where the Büchi automata are 1-weak.

1998 ACM Subject Classification F.1.1; F.1.2; F.2.2

Keywords and phrases Concurrent games, Nash equilibria, Büchi objectives

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.375

1 Introduction

Game theory (especially games played on graphs) is used in computer science as a powerful framework for modelling interactions in embedded systems [18, 13]. Until recently, more focus had been put on purely antagonistic situations, where the system should fulfil its specification however the environment behaves. This situation can be modelled as a two-player game (one player for the system, and one for the environment), and a winning strategy for the first player is a good controller for the system. In this purely antagonistic view, the objectives of both players are opposite: the aim of the second player is to prevent the first player from achieving her own objective; such games are called zero-sum.

In many cases, however, games are non-zero-sum, especially when they involve more than two players. Such games appear e.g. in various problems in telecommunications, where several agents try to send data on a network [10]. Focusing only on winning strategies in this setting may then be too narrow: winning strategies must be winning against any behaviour of the other agents, and do not consider the fact that the other agents also have their own objectives. In the non-zero-sum setting, each player can have a different payoff associated with an outcome of the game; it is then more interesting to look for *equilibria*. For instance, a Nash equilibrium is a behaviour of the agents in which they play rationally, in the sense that no agent can get a better payoff by unilaterally switching to another strategy [15]. This corresponds to stable states of the game. Note that Nash equilibria need not exist and are not necessarily *optimal*: several equilibria can coexist, possibly with different payoffs.

Our contribution. We focus here on qualitative objectives for the players: such objectives are ω-regular properties over infinite plays, and a player receives payoff 1 if the property is fulfilled and 0 otherwise. Our aim is to decide the existence of pure-strategy Nash equilibria in nondeterministic concurrent games. Being concurrent (instead of the more classical *turn-based*

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 375–386

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

games) and nondeterministic are two important properties of *timed games* (which are games played on timed automata [2, 9]), to which we ultimately want to apply our algorithms.

In a first part, we focus on internal Büchi conditions (defined on the game directly) and show that we can decide the existence of equilibria in polynomial time, which has to be compared with the NP-completeness of the problem in the case of reachability objectives [8, 3]. This relies on an iterated version of a *repellor operator* [3]. Roughly speaking, the repellor is to the computation of Nash equilibria in non-zero-sum games what the attractor is to the computation of winning states in zero-sum games. The repellor operator we use for Büchi objectives is a generalisation of the one defined in [3] for reachability objectives, and the proof techniques are more involved.

Then, using a simulation lemma, we show how to compute Nash equilibria in case the objectives of the players are given by deterministic Büchi automata. This encompasses many winning conditions (among which reachability, Büchi, safety, etc.), and we show that deciding the existence of Nash equilibria with constraints on the payoff is EXPTIME-complete. Under a certain restriction on the automata (1-weakness), we prove that the complexity reduces to PSPACE, and we prove PSPACE-hardness for the special case of safety objectives. When the game is *deterministic* and the 1-weak Büchi automata defining the winning conditions have bounded size (this includes safety and reachability objectives), we show that the constrained existence problem becomes NP-complete. The simulation lemma can also be used to lift our results to timed games, for which all our problems are EXPTIME-complete.

Related work. Concurrent and, more generally, stochastic games go back to Shapley [17]. However, most research in game theory and economics has focused on games with rewards, which are either averaged or discounted along an infinite path. In particular, Fink [11] proved that every discounted stochastic game has a Nash equilibrium in pure strategies, and Vielle [22] proved the existence of ϵ-*equilibria* in randomised strategies for two-player stochastic games under the average-reward criterion. For two-player concurrent games with Büchi objectives, the existence of ϵ-equilibria (in randomised strategies) was proved by Chatterjee [5]. However, exact Nash equilibria need not exist. An important subclass where even Nash equilibria in pure strategies exist are turn-based games with Büchi objectives [8].

The complexity of Nash equilibria in games played on graphs was first addressed in [8, 19]. In particular, it was shown in [19] that the existence of a Nash equilibrium with a constraint on its payoff can be decided in polynomial time for turn-based games with Büchi objectives. In this paper, we extend this result to concurrent games. It was also shown in [19] that the same problem is NP-hard for turn-based games with co-Büchi conditions, which implies hardness for concurrent games with this kind of objectives. For concurrent games with ω-regular objectives, the decidability of the constrained existence problem w.r.t. pure strategies was established by Fisman et al. [12], but their algorithm runs in doubly exponential time, whereas our algorithm for Büchi games runs in polynomial time. Finally, Ummels and Wojtczak [21] proved that the existence of a Nash equilibrium in pure or randomised strategies is undecidable for *stochastic* games with reachability or Büchi objectives, which justifies our restriction to concurrent games without probabilistic transitions (see [20] for a similar undecidability result for randomised Nash equilibria in non-stochastic games).

2 Preliminaries

2.1 Concurrent Games

A *transition system* is a 2-tuple $\mathcal{S} = \langle \text{States}, \text{Edg} \rangle$ where States is a (possibly uncountable) set of states and $\text{Edg} \subseteq \text{States} \times \text{States}$ is the set of transitions. In a transition system $\mathcal{S}$, a *path* π is a non-empty sequence $(s_i)_{0 \le i < n}$ (where $n \in \mathbb{N} \cup \{\infty\}$) of states such that $(s_i, s_{i+1}) \in \text{Edg}$ for all $i < n-1$. The *length* of π, denoted by $|\pi|$, is $n-1$. The set of finite paths (also called *histories*) of $\mathcal{S}$ is denoted by $\text{Hist}_{\mathcal{S}}$, the set of infinite paths (also called *plays*) of $\mathcal{S}$ is denoted by $\text{Play}_{\mathcal{S}}$, and $\text{Path}_{\mathcal{S}} = \text{Hist}_{\mathcal{S}} \cup \text{Play}_{\mathcal{S}}$ is the set of paths of $\mathcal{S}$. Given a path $\pi = (s_i)_{0 \le i < n}$ and an integer $j < n$, the *j-th prefix* (resp. *j-th suffix*, *j-th state*) of π, denoted by $\pi_{\le j}$ (resp. $\pi_{\ge j}$, $\pi_{=j}$), is the finite path $(s_i)_{0 \le i < j+1}$ (resp. $(s_i)_{j \le i < n}$, state s_j). If $\pi = (s_i)_{0 \le i < n}$ is a history, we write $\text{last}(\pi) = s_{|\pi|}$.

We consider nondeterministic concurrent games [3], which extend standard concurrent games [1] with nondeterminism.

▶ **Definition 1.** A *(nondeterministic) concurrent game* is a tuple $\mathcal{G} = \langle \text{States}, \text{Edg}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab}, (\mathcal{L}_A)_{A \in \text{Agt}} \rangle$, where $\langle \text{States}, \text{Edg} \rangle$ is a transition system, Agt is a finite set of players, Act is a (possibly uncountable) set of actions, and

- $\text{Mov}\colon \text{States} \times \text{Agt} \to 2^{\text{Act}} \setminus \{\varnothing\}$ is a mapping indicating the actions available to a given player in a given state;
- $\text{Tab}\colon \text{States} \times \text{Act}^{\text{Agt}} \to 2^{\text{Edg}} \setminus \{\varnothing\}$ associates with a state and an action profile the resulting set of edges; we require that $s = s'$ if $(s', s'') \in \text{Tab}(s, \langle m_A \rangle_{A \in \text{Agt}})$;
- $\mathcal{L}_A \subseteq \text{States}^\omega$ defines the *objective* for player $A \in \text{Agt}$; the *payoff* for player A is the function $\nu_A\colon \text{States}^\omega \to \{0, 1\}$, where $\nu_A(\pi) = 1$ if $\pi \in \mathcal{L}_A$, and $\nu_A(\pi) = 0$ otherwise; we say that player A *prefers* play π' over play π, denoted $\pi \preccurlyeq_A \pi'$, if $\nu_A(\pi) \le \nu_A(\pi')$.

We call a game $\mathcal{G}$ *finite* if its set of states is finite.

Non-determinism naturally appears in timed games, and this is our most important motivation for investigating this extension of standard concurrent games. We explain in Section 4.3 how our results apply to timed games.

We say that a *move* $\langle m_A \rangle_{A \in \text{Agt}} \in \text{Act}^{\text{Agt}}$ (which we may write m_{Agt} in the sequel) is *legal* at s if $m_A \in \text{Mov}(s, A)$ for all $A \in \text{Agt}$. A concurrent game is *deterministic* if $\text{Tab}(s, m_{\text{Agt}})$ is a singleton for each $s \in \text{States}$ and each legal move m_{Agt} (at s). A game is *turn-based* if for each state the set of allowed moves is a singleton for all but at most one player.

In a nondeterministic concurrent game, whenever we arrive at a state s, the players (simultaneously) choose a legal move m_{Agt}. Then, one of the transitions in $\text{Tab}(s, m_{\text{Agt}})$ is nondeterministically selected, which results in a new state of the game. In the sequel, we write $\text{Hist}_{\mathcal{G}}$, $\text{Play}_{\mathcal{G}}$ and $\text{Path}_{\mathcal{G}}$ for the corresponding set of paths in the underlying transition system of $\mathcal{G}$. We also write $\text{Hist}_{\mathcal{G}}(s)$, $\text{Play}_{\mathcal{G}}(s)$ and $\text{Path}_{\mathcal{G}}(s)$ for the respective subsets of paths starting in state s.

▶ **Definition 2.** Let $\mathcal{G}$ be a concurrent game, and $A \in \text{Agt}$. A *strategy* for A is a mapping $\sigma_A\colon \text{Hist}_{\mathcal{G}} \to \text{Act}$ such that $\sigma_A(\pi) \in \text{Mov}(\text{last}(\pi), A)$ for all $\pi \in \text{Hist}_{\mathcal{G}}$. A strategy σ_P for a coalition P is a tuple of strategies, one for each player in P. We write $\sigma_P = \langle \sigma_A \rangle_{A \in P}$ for such a strategy. A *strategy profile* is a strategy for the coalition Agt. We write $\text{Strat}_{\mathcal{G}}^P$ for the set of strategies of coalition P (or simply $\text{Strat}_{\mathcal{G}}^B$ if $P = \{B\}$), and $\text{Prof}_{\mathcal{G}} = \text{Strat}_{\mathcal{G}}^{\text{Agt}}$.

Note that we only consider non-randomised (*pure*) strategies in this paper. Notice also that strategies are based on the sequences of visited states, and not on the actions played by

the players. This is a realistic assumption for concurrent systems where different components interact with each other and each component has a set of internal actions which cannot be observed by the other components. However, it makes the computation of equilibria harder: when a deviation from the equilibrium profile occurs, the given sequence of states does not uniquely determine the player who has deviated. While the main part of the paper focuses on state-based strategies for Büchi objectives, we show in Section 6 that equilibria in the action-based setting can be computed more easily, even for parity objectives.

Let $\mathcal{G}$ be a game, P a coalition, and σ_P a strategy for P. A path $\pi = (s_j)_{0 \leq j \leq |\pi|}$ is *compatible* with the strategy σ_P if, for all $k < |\pi|$, there exists a move m_{Agt} such that (i) m_{Agt} is legal at s_k, (ii) $m_A = \sigma_A(\pi_{\leq k})$ for all $A \in P$, and (iii). $(s_k, s_{k+1}) \in \mathrm{Tab}(s_k, m_{\mathrm{Agt}})$. We write $\mathrm{Out}_{\mathcal{G}}(\sigma_P)$ for the set of paths (or *outcomes*) in $\mathcal{G}$ that are compatible with the strategy σ_P, and we write $\mathrm{Out}^{\mathrm{f}}_{\mathcal{G}}(\sigma_P)$ (resp. $\mathrm{Out}^{\infty}_{\mathcal{G}}(\sigma_P)$) for the finite (resp. infinite) outcomes, and $\mathrm{Out}_{\mathcal{G}}(s, \sigma_P)$, $\mathrm{Out}^{\mathrm{f}}_{\mathcal{G}}(s, \sigma_P)$ and $\mathrm{Out}^{\infty}_{\mathcal{G}}(s, \sigma_P)$ for the respective sets of outcomes that start in state s. In general there might be several infinite outcomes for a strategy profile from a given state. However, in the case of deterministic games, any strategy profile has a single infinite outcome from a given state.

▶ **Example 3.** Figure 1 depicts a two-player concurrent game, called the *matching-penny* game. A pair $\langle a, b \rangle$ represents a move, where Player 1 plays action a and Player 2 plays b. Starting from state s_0, if both players choose the same action, then the game proceeds to s_1; otherwise, the game proceeds to s_2. In the matching-penny game, the objective for Player 1 is to visit s_1 (which is encoded as $\mathcal{L}_1 = \mathrm{States}^* \cdot \{s_1\} \cdot \mathrm{States}^\omega$), while for Player 2 it is to visit s_2. Figure 2 shows another game (our running example), in which the objective of Player 1 is to *loop* in s_1 ($\mathcal{L}_1 = \mathrm{States}^* \cdot \{s_1\}^\omega$), whereas the objective for Player 2 is to *loop* in s_2 ($\mathcal{L}_2 = \mathrm{States}^* \cdot \{s_2\}^\omega$).

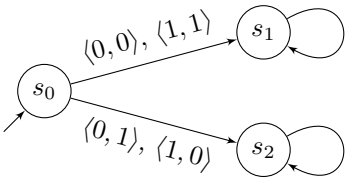


Figure 1 The matching-penny game

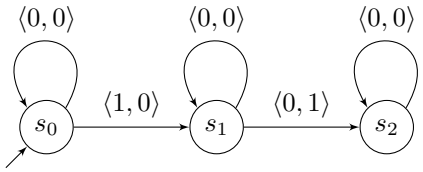


Figure 2 A game with Büchi objectives

2.2 Pseudo-Nash Equilibria

Given a move m_{Agt} and an action m' for some player B, we write $m_{\mathrm{Agt}}[B \mapsto m']$ for the move n_{Agt} with $n_A = m_A$ if $A \neq B$ and $n_B = m'$. This is extended to strategies in the natural way. For non-zero-sum games, several notions of equilibria have been defined, e.g. Nash equilibria [15], subgame-perfect equilibria [16], and secure equilibria [6]. None of these notions apply to nondeterministic games. Bouyer et al. have therefore proposed the notion of *pseudo-Nash equilibria* [3], which extend standard Nash equilibria to nondeterministic games.

▶ **Definition 4.** Let $\mathcal{G}$ be a nondeterministic concurrent game with objectives $(\mathcal{L}_A)_{A \in \mathrm{Agt}}$, and let s be a state of $\mathcal{G}$. A *pseudo-Nash equilibrium* of $\mathcal{G}$ from s is a pair $(\sigma_{\mathrm{Agt}}, \pi)$ of a strategy profile $\sigma_{\mathrm{Agt}} \in \mathrm{Prof}_{\mathcal{G}}$ and a play $\pi \in \mathrm{Out}_{\mathcal{G}}(s, \sigma_{\mathrm{Agt}})$ such that $\pi' \preccurlyeq_B \pi$ for all players $B \in \mathrm{Agt}$, all strategies $\sigma' \in \mathrm{Strat}^B$, and all plays $\pi' \in \mathrm{Out}_{\mathcal{G}}(s, \sigma_{\mathrm{Agt}}[B \mapsto \sigma'])$. The outcome π is then called an *optimal play* for the strategy profile σ_{Agt}.

For deterministic games, the play π is uniquely determined by σ_{Agt}, so that pseudo-Nash equilibria coincide with *Nash equilibria* [15]: these are strategy profiles where no player has an incentive to unilaterally deviate from her strategy.

In the case of nondeterministic games, a strategy profile for an equilibrium may give rise to several outcomes. The outcome π is then chosen cooperatively by all players: once a strategy profile is fixed, nondeterminism is resolved by all players choosing one of the possible outcomes in such a way that each player has no incentive to unilaterally changing her choice (nor her strategy). To the best of our knowledge, this cannot be encoded by adding an extra player for resolving the nondeterminism.

▶ **Example 5.** Clearly, there is no pure-strategy Nash equilibrium in the game of Figure 1 since a losing player can always improve her payoff by switching her choice. In the game of Figure 2 (where Player i wants to visit s_i infinitely often), there are several Nash equilibria: one with payoff $(0, 1)$, in which both players play action 1 when it is available; another one with payoff $(0, 0)$, in which Player 1 always plays 0, and Player 2 plays 1 when available.

In this paper, we study several decision problems related to the existence of pseudo-Nash equilibria. The *existence problem* consists in deciding the existence of a pseudo-Nash equilibrium in a given state of a game. Since several pseudo-Nash equilibria may coexist, it is also interesting to decide whether there is one with a given payoff (0 or 1) for some of the players; this is the *constrained existence problem*. Finally, the *verification problem* asks whether a given payoff function (totally defined over Agt) is the payoff of some pseudo-Nash equilibrium. Notice that the first and third problems are trivially logspace-reducible to the second one.

3 Internal Büchi Objectives

In this section, we fix a nondeterministic concurrent game $\mathcal{G} = \langle \mathrm{States}, \mathrm{Edg}, \mathrm{Agt}, \mathrm{Act}, \mathrm{Mov}, \mathrm{Tab}, (\mathcal{L}_A)_{A\in\mathrm{Agt}}\rangle$, where the objectives are *internal* Büchi conditions given by a set $\Omega_A \subseteq \mathrm{States}$ of target states for each player $A \in \mathrm{Agt}$. The corresponding objective for player A is the set $\mathcal{L}_A = \{\pi \in \mathrm{States}^\omega \mid \pi_{=j} \in \Omega_A \text{ for infinitely many } j \in \mathbb{N}\}$.

3.1 Characterising Equilibria Using Fixpoints

In [3], pseudo-Nash equilibria are characterised for qualitative reachability objectives using a fixpoint computation called the *repellor*. This was the counter-part of the attractor in non-zero-sum games for computing equilibria. In this section, we extend the repellor to handle internal Büchi objectives.

Suspect players. Let $e = (s, s')$ be an edge. Given a move m_{Agt}, we define the set of suspect players for e as the set

$$\mathrm{Susp}(e, m_{\mathrm{Agt}}) = \{B \in \mathrm{Agt} \mid \exists\, m' \in \mathrm{Mov}(s, B) \text{ such that } e \in \mathrm{Tab}(s, m_{\mathrm{Agt}}[B \mapsto m'])\}.$$

Intuitively, Player $B \in \mathrm{Agt}$ is a suspect for edge e and if she can unilaterally change her action to trigger edge e. Notice that if $e \in \mathrm{Tab}(s, m_{\mathrm{Agt}})$, then $\mathrm{Susp}(e, m_{\mathrm{Agt}}) = \mathrm{Agt}$.

The iterated (or Büchi) repellor. For any $n \in \mathbb{N}$ and $P \subseteq \mathrm{Agt}$, we define the *n-th repellor set* $\mathrm{Rep}^n_{\mathcal{G}}(P)$ as follows. If $n = 0$, then $\mathrm{Rep}^0_{\mathcal{G}}(P) = \emptyset$ for any $P \subseteq \mathrm{Agt}$. Now fix $n \in \mathbb{N}$, and assume that repellor sets $\mathrm{Rep}^n_{\mathcal{G}}(P)$ have been defined for any $P \subseteq \mathrm{Agt}$. As the base case for level $n + 1$, we set $\mathrm{Rep}^{n+1}_{\mathcal{G}}(\emptyset) = \mathrm{States}$. Then, assuming that $\mathrm{Rep}^{n+1}_{\mathcal{G}}(P')$ has been defined

for all $P' \subsetneq P$, we let $\mathrm{Rep}_{\mathcal{G}}^{n+1}(P)$ be the largest set fulfilling the following condition: for all $s \in \mathrm{Rep}_{\mathcal{G}}^{n+1}(P)$ there exists a legal move m_{Agt} (at s) such that

1. $s' \in \mathrm{Rep}_{\mathcal{G}}^{n+1}(P \cap \mathrm{Susp}_{\mathcal{G}}((s,s'), m_{\mathrm{Agt}}))$ for all $s' \in \mathrm{States}$, and
2. if $s' \in \Omega_A$ for some player $A \in P \cap \mathrm{Susp}_{\mathcal{G}}((s,s'), m_{\mathrm{Agt}})$, then $s' \in \mathrm{Rep}_{\mathcal{G}}^{n}(P \cap \mathrm{Susp}_{\mathcal{G}}((s,s'), m_{\mathrm{Agt}}))$.

Given a state $s \in \mathrm{Rep}_{\mathcal{G}}^{n+1}(P)$, a legal move m_{Agt} that fulfils 1. and 2. is called a *secure move* (w.r.t. P and $n+1$); we write $\mathrm{Secure}_{\mathcal{G}}^{n+1}(s,P)$ for the set of these moves. Finally, we define $\mathrm{Rep}_{\mathcal{G}}^{\infty}(P) = \bigcap_{n \geq 0} \mathrm{Rep}^n(P)$. In the following, to improve readability, we will omit the index $\mathcal{G}$ in all the notions we have defined, when the game is clear from the context.

Intuitively, a state s is an element of $\mathrm{Rep}_{\mathcal{G}}^{n}(P)$ iff at s there is a legal move such that no player $A \in P$ can force to visit her set of target states at least n times by changing her action. For finite games, it follows that a state s is an element of $\mathrm{Rep}_{\mathcal{G}}^{\infty}(P)$ iff at s there is a legal move such that no player $A \in P$ can force to visit her set set of target states *infinitely often* by changing her action.

▶ Remark. The repellor defined for reachability objectives in [3] is rather similar to $\mathrm{Rep}^1(P)$; it differs only in the second condition, which was "$\mathrm{Rep}^1(P) \cap \Omega_A = \emptyset$ for all $A \in P$" in [3]. This change is required since a play that is losing w.r.t. a Büchi objective might visit a winning state a finite number of times (whereas this cannot happen for reachability objectives).

▶ **Example 6.** In the game of Figure 2, if we assume reachability objectives (state s_i for Player i), there is no equilibrium with payoff $(0,0)$, since Player 1 can enforce a visit to her winning state. If we assume Büchi objectives, we have seen in Example 5 that there is an equilibrium with payoff $(0,0)$. Table 1 displays the values of the iterated repellors in this game, for all possible sets of players. These results were obtained with our prototype implementation of our algorithms, available at `http://www.lsv.ens-cachan.fr/Software/praline/`.

Table 1 Computing the repellor sets in the game of Figure 2

P	$\mathrm{Rep}^0(P)$	$\mathrm{Rep}^1(P)$	$\mathrm{Rep}^2(P)$	$\mathrm{Rep}^\infty(P) = \mathrm{Rep}^3(P)$
$\emptyset$	$\emptyset$	$\{s_0, s_1, s_2\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_1, s_2\}$
$\{A_1\}$	$\emptyset$	$\{s_1, s_2\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_1, s_2\}$
$\{A_2\}$	$\emptyset$	$\{s_0\}$	$\{s_0\}$	$\{s_0\}$
$\{A_1, A_2\}$	$\emptyset$	$\emptyset$	$\{s_0\}$	$\{s_0\}$

▶ **Lemma 7.** *The repellor and the secure moves satisfy the following properties:*

- *If* $P' \subseteq P \subseteq \mathrm{Agt}$, *then* $\mathrm{Rep}^n(P) \subseteq \mathrm{Rep}^n(P')$ *for all* $n \in \mathbb{N}$.
- $\mathrm{Rep}^n(P) \subseteq \mathrm{Rep}^{n+1}(P)$ *for all* $P \subseteq \mathrm{Agt}$ *and* $n \in \mathbb{N}$.
- $\mathrm{Secure}^n(s,P) \subseteq \mathrm{Secure}^{n+1}(s,P)$ *for all* $P \subseteq \mathrm{Agt}$, $n \in \mathbb{N}$ *and* $s \in \mathrm{States}$.

We define the *n-th repellor transition system* $\mathcal{S}^n(P) = \langle \mathrm{States}, \mathrm{Edg}_n \rangle$ by $(s,s') \in \mathrm{Edg}_n$ iff there exists $m_{\mathrm{Agt}} \in \mathrm{Secure}^n(s,P)$ such that $(s,s') \in \mathrm{Tab}(s, m_{\mathrm{Agt}})$. Note in particular that any $s \in \mathrm{Rep}^n(P)$ has an outgoing transition in $\mathcal{S}^n(P)$. We also define the limit repellor transition system $\mathcal{S}^\infty(P) = \langle \mathrm{States}, \bigcap_{n \geq 0} \mathrm{Edg}_n \rangle$. The following lemma bounds the number of iteration steps required to reach $\mathrm{Rep}^\infty(P)$.

▶ **Lemma 8.** *Let* $\mathcal{G}$ *be a a finite game,* $P \subseteq \mathrm{Agt}$, *and let* ℓ *be the length of the longest acyclic path in* $\mathcal{G}$. *Then* $\mathrm{Rep}^n(P) = \mathrm{Rep}^\infty(P)$ *for all* $n \geq \ell \cdot |P|$.

The correctness of the iterated repellor for *finite* games is stated in the next proposition.

▶ **Proposition 9.** *Let $\mathcal{G}$ be a finite game, $P \subseteq \mathrm{Agt}$, and let $\rho \in \mathrm{Play}(s)$ be a play that visits $\bigcup_{B\in P} \Omega_B$ only finitely often. Then ρ is a path in $\mathcal{S}^\infty(P)$ if and only if there exists $\sigma_{\mathrm{Agt}} \in \mathrm{Prof}$ such that $\rho \in \mathrm{Out}(s, \sigma_{\mathrm{Agt}})$ and ρ' does not visit Ω_B infinitely often for all plays ρ' that can arise when some player $B \in P$ changes her strategy, i.e. $\rho' \in \mathrm{Out}(s, \sigma_{\mathrm{Agt}}[B \mapsto \sigma'])$ for some $B \in P$ and some $\sigma' \in \mathrm{Strat}^B$.*

We can deduce from this proposition that if ρ is an infinite path from state s in $\mathcal{S}^\infty(P)$ that visits Ω_A infinitely often *if and only if* $A \notin P$, then there is a pseudo Nash equilibrium from s with optimal play ρ.

▶ **Corollary 10.** *Let $\mathcal{G}$ be a finite game, $s \in \mathrm{States}$, and $\nu\colon \mathrm{Agt} \to \{0,1\}$. There exists a pseudo-Nash equilibrium in $\mathcal{G}$ with payoff ν if and only if there exists an infinite path ρ in $\mathcal{S}^\infty(\nu^{-1}(0))$ with payoff $\nu_A(\rho) = \nu(A)$ for all $A \in \mathrm{Agt}$.*

3.2 Application to Solving the Three Problems

We use the previous characterisation for analysing the complexity of the various decision problems that we have defined in Section 2.2.

▶ **Theorem 11.** *The verification, existence and constrained existence problems for finite games with internal Büchi objectives are* PTIME*-complete.*

The lower bounds are simple adaptations of the PTIME-hardness of the circuit value problem. We now focus on the PTIME upper bounds, and prove it for the constrained existence problem (which implies the same upper bound for the other two problems).

We first use the equivalence given in Proposition 9 to get a set-based characterisation of (pseudo-)Nash equilibria. We fix a set of winning players $W \subseteq \mathrm{Agt}$ and a set of losing players $L \subseteq \mathrm{Agt}$, and we fix an initial state s. Given a transition system $\langle S, E\rangle$ and a set of players P, we say that they satisfy condition (‡) if the following properties are fulfilled:

(1) $\Omega_A \cap S = \emptyset$ if and only if $A \in P$;
(2) $L \subseteq P$ and $P \cap W = \emptyset$;
(3) $\langle S, E\rangle$ is strongly connected;
(4) $\langle S, E\rangle \subseteq \mathcal{S}^\infty(P)$;
(5) S is reachable from s in $\mathcal{S}^\infty(P)$.

The following is then a corollary to Proposition 9.

▶ **Corollary 12** (Set-based characterisation)**.** *A pair $(\langle S, E\rangle, P)$ satisfies condition (‡) iff there is an infinite path ρ in $\mathcal{S}^\infty(P)$ from s that, from some point onwards, stays in $\langle S, E\rangle$. In particular, ρ is losing for all players in L. Moreover, if $(\langle S, E\rangle, P)$ satisfies (‡), then there exists an infinite path ρ in $\mathcal{S}^\infty(P)$ from s with the same property that visits* all *states of S infinitely often (and is thus winning for all players in W).*

Note that in the above characterisation, P is uniquely determined by the set S; hence we write $P(S) = \{A \in \mathrm{Agt} \mid S \cap \Omega_A = \emptyset\}$, and we say that $\langle S, E\rangle$ satisfies condition (‡) if $(\langle S, E\rangle, P(S))$ does. Our aim is to compute in polynomial time all *maximal* pairs $\langle S, E\rangle$ that satisfy condition (‡). As a prerequisite, we assume that we can compute $\mathcal{S}^\infty(P)$ in polynomial time whenever $P \subseteq \mathrm{Agt}$ is given. This can be proved using similar arguments as in [4]. Now, we define a recursive operator SSG (SolveSubGame) by setting $\mathsf{SSG}(\langle S, E\rangle) = \{\langle S, E\rangle\}$ if $\langle S, E\rangle \subseteq \mathcal{S}^\infty(P(S))$ and $\langle S, E\rangle$ is strongly connected, and

$$\mathsf{SSG}(\langle S, E\rangle) = \bigcup_{\langle S', E'\rangle \in \mathrm{SCC}(\langle S, E\rangle)} \mathsf{SSG}(\langle S', E'\rangle \cap \mathcal{S}^\infty(P(S')))$$

in all other cases. Here, $\mathrm{SCC}(\langle S, E\rangle)$ denotes the set of strongly connected components of $\langle S, E\rangle$ (which can be computed in linear time). Finally, we define

$$\mathsf{Sol}(L, W) = \mathsf{SSG}\Big(\langle \mathrm{States} \setminus \bigcup_{A \in L} \Omega_A, \mathrm{Edg}\rangle\Big) \cap \{\langle S, E\rangle \mid P(S) \cap W = \emptyset\}\,.$$

▶ **Lemma 13.** *If $\langle S, E\rangle \in \mathsf{Sol}(L, W)$ and S is reachable from s in $\mathcal{S}^\infty(P(S))$, then it satisfies condition (‡). Conversely, if $\langle S, E\rangle$ satisfies condition (‡), then there exists $\langle S', E'\rangle \in \mathsf{Sol}(L, W)$ such that $\langle S, E\rangle \subseteq \langle S', E'\rangle$.*

▶ **Lemma 14.** *The set $\mathsf{Sol}(L, W)$ can be computed in polynomial time.*

The PTIME upper bound of Theorem 11 follows from the above analysis.

▶ Remark. This result may seem surprising since we know that the problem is NP-complete for reachability objectives, even in turn-based games [8, 3]. Intuitively, the problem is harder for reachability objectives because whether a play satisfies or not a reachability objective is not only determined by its behaviour in the strongly connected component in which it settles but on *all* visited states.

4 Game Simulations

Our aim is to transfer our results for internal Büchi objectives to larger classes of objectives. A useful tool is the notion of game simulation, which we develop now.

4.1 Definition and General Properties

We already gave a definition of game simulation in [3], which was tailored to games with reachability objectives; we extend this notion to games with arbitrary qualitative objectives.

▶ **Definition 15.** Consider two games $\mathcal{G} = \langle \mathrm{States}, \mathrm{Edg}, \mathrm{Agt}, \mathrm{Act}, \mathrm{Mov}, \mathrm{Tab}, (\mathcal{L}_A)_{A \in \mathrm{Agt}}\rangle$ and $\mathcal{G}' = \langle \mathrm{States}', \mathrm{Edg}', \mathrm{Agt}, \mathrm{Act}', \mathrm{Mov}', \mathrm{Tab}', (\mathcal{L}'_A)_{A \in \mathrm{Agt}}\rangle$ with the same set Agt of players. A relation $\lhd \subseteq \mathrm{States} \times \mathrm{States}'$ is a *game simulation* if $s \lhd s'$ implies that for each move m_{Agt} in $\mathcal{G}$ there exists a move m'_{Agt} in $\mathcal{G}'$ such that

1. for each $t' \in \mathrm{States}'$ there exists $t \in \mathrm{States}$ with $t \lhd t'$ and $\mathrm{Susp}((s', t'), m'_{\mathrm{Agt}}) \subseteq \mathrm{Susp}((s, t), m_{\mathrm{Agt}})$, and
2. for each $(s, t) \in \mathrm{Tab}(s, m_{\mathrm{Agt}})$ there exists $(s', t') \in \mathrm{Tab}'(s', m'_{\mathrm{Agt}})$ with $t \lhd t'$.

If $\lhd$ is a game simulation, we say that $\mathcal{G}'$ *simulates* $\mathcal{G}$. Finally, a game simulation $\lhd$ is *winning-preserving* from $(s_0, s'_0) \in \mathrm{States} \times \mathrm{States}'$ if for all $\rho \in \mathrm{Play}_{\mathcal{G}}(s_0)$ and $\rho' \in \mathrm{Play}_{\mathcal{G}'}(s'_0)$ with $\rho \lhd \rho'$ (i.e. $\rho_{=i} \lhd \rho'_{=i}$ for all $i \in \mathbb{N}$) it holds that $\rho \in \mathcal{L}_A$ iff $\rho' \in \mathcal{L}'_A$ for all $A \in \mathrm{Agt}$.

▶ **Proposition 16.** *Game simulation is transitive.*

▶ **Proposition 17.** *Assume $\mathcal{G}$ and $\mathcal{G}'$ are games. Fix two states s and s' in $\mathcal{G}$ and $\mathcal{G}'$ respectively, and assume that $\lhd$ is a winning-preserving game simulation from (s, s'). If there exists a pseudo-Nash equilibrium $(\sigma_{\mathrm{Agt}}, \rho)$ of $\mathcal{G}$ from s, then there exists a pseudo-Nash equilibrium $(\sigma'_{\mathrm{Agt}}, \rho')$ of $\mathcal{G}'$ from s' with $\rho \lhd \rho'$. In particular, ρ and ρ' have the same payoff.*

4.2 Product of a Game with Deterministic Büchi Automata

We use the results on game simulation to study objectives that are defined by deterministic Büchi automata. A *deterministic Büchi automaton* $\mathcal{A}$ over alphabet Σ is a tuple $\langle Q, \Sigma, \delta, q_0, R\rangle$, where Q is a finite set of *states*, Σ is the *input alphabet*, $\delta\colon Q \times \Sigma \to Q$ is the *transition function*, $q_0 \in Q$ is the *initial* state, and $R \subseteq Q$ is the set of *repeated* states. We assume that the reader is familiar with Büchi automata, and we write $\mathrm{L}(\mathcal{A}) \subseteq \Sigma^\omega$ for the language accepted by $\mathcal{A}$.

Fix a game $\mathcal{G} = \langle \mathrm{States}, \mathrm{Edg}, \mathrm{Agt}, \mathrm{Act}, \mathrm{Mov}, \mathrm{Tab}, (\mathcal{L}_B)_{B\in\mathrm{Agt}}\rangle$ and a player $A \in \mathrm{Agt}$, and assume that $\mathcal{L}_A = \mathrm{L}(\mathcal{A})$ for a deterministic Büchi automaton $\mathcal{A} = \langle Q, \mathrm{States}, \delta, q_0, R\rangle$ over States. We show how to compute pseudo-Nash equilibria in $\mathcal{G}$ by building a product of $\mathcal{G}$ with $\mathcal{A}$ and computing the pseudo-Nash equilibria in the resulting game.

We define the product of the game $\mathcal{G}$ with the automaton $\mathcal{A}$ as the game $\mathcal{G} \ltimes \mathcal{A} = \langle \mathrm{States}', \mathrm{Edg}', \mathrm{Agt}, \mathrm{Act}, \mathrm{Mov}', \mathrm{Tab}', (\mathcal{L}'_B)_{B\in\mathrm{Agt}}\rangle$, where:

- $\mathrm{States}' = \mathrm{States} \times Q$;
- $\mathrm{Edg}' = \{((s,q),(s',q')) \mid (s,s') \in \mathrm{Edg} \text{ and } \delta(q,s) = q'\}$;
- $\mathrm{Mov}'((s,q),A_i) = \mathrm{Mov}(s,A_i)$ for every $A_i \in \mathrm{Agt}$;
- $\mathrm{Tab}'((s,q),m_{\mathrm{Agt}}) = \{((s,q),(s',q')) \mid (s,s') \in \mathrm{Tab}(s,m_{\mathrm{Agt}}) \text{ and } \delta(q,s) = q'\}$;
- if $B = A$, then $\mathcal{L}'_B$ is the internal Büchi objective given by the set $\Omega = \mathrm{States} \times R$; otherwise, $\mathcal{L}'_B = \pi^{-1}(\mathcal{L}_B)$, where π is the natural projection from States' to States and its extension to plays (i.e. $\pi((s_0,q_0)(s_1,q_1)\dots) = s_0 s_1 \dots$).

▶ Remark. Note that, if $\mathcal{L}_B$ is defined by an internal Büchi condition, then so is $\mathcal{L}'_B$.

▶ **Lemma 18.** *$\mathcal{G} \ltimes \mathcal{A}$ simulates $\mathcal{G}$, and vice versa. Furthermore, in both cases we can exhibit a game-simulation that is winning-preserving from $(s,(s,q_0))$ for all $s \in \mathrm{States}$.*

Assume that for each player $A_i \in \mathrm{Agt}$ the objective in $\mathcal{G}$ is given by a deterministic Büchi automaton $\mathcal{A}_i$. We use the transitivity of game simulation to build a product of $\mathcal{G}$ with each of the automata $\mathcal{A}_i$, namely $\mathcal{G}' = \mathcal{G} \ltimes \mathcal{A}_1 \ltimes \cdots \ltimes \mathcal{A}_n$ (we assume that $\ltimes$ is left-associative). Each player $A_i \in \mathrm{Agt}$ has an internal Büchi objective in $\mathcal{G}'$, which we denote by Ω_i.

▶ **Corollary 19.** *Let $s \in \mathrm{States}$ and $\nu\colon \mathrm{Agt} \to \{0,1\}$. There is a pseudo-Nash equilibrium in $\mathcal{G}$ from s with payoff ν if and only if there is a pseudo-Nash equilibrium in $\mathcal{G}'$ from $(s, q_{01}, \dots, q_{0n})$ with payoff ν, where q_{0i} is the initial state of $\mathcal{A}_i$.*

4.3 Application to Timed Games

We now apply the game-simulation approach to the computation of pseudo-Nash equilibria in timed games. Given a timed game $\mathcal{G}$ with internal Büchi objectives, the corresponding (exponential-size) region game $\mathcal{R}_\mathcal{G}$ as defined in [4] simulates $\mathcal{G}$ and is simulated by $\mathcal{G}$ while preserving winning conditions (the proof for reachability objectives in [4] can be easily extended to our framework). Pseudo-Nash equilibria of $\mathcal{G}$ can thus be computed on the finite game $\mathcal{R}_\mathcal{G}$. If the objectives of the players are defined by deterministic Büchi automata $(\mathcal{A}_i)_{A_i\in\mathrm{Agt}}$, we can compute the product $\mathcal{R}_\mathcal{G} \ltimes \mathcal{A}_1 \ltimes \cdots \ltimes \mathcal{A}_n$ with corresponding internal Büchi objectives $(\Omega_i)_{A_i\in\mathrm{Agt}}$, as defined in the previous subsection. This product has size exponential in the size of $\mathcal{G}$ and in the number of players. We can then apply the algorithm developed in Section 3.2, yielding an EXPTIME upper bound for deciding the verification, existence, and constrained existence problems in timed games. Finally we get EXPTIME-hardness for internal Büchi objectives by applying the reduction in [4, Prop. 20] (replace all accepting sink states by repeated states).

▶ **Theorem 20.** *The verification, existence, and constrained existence problems are* EXPTIME*-complete both for timed games with internal Büchi objectives and for timed games with objectives defined by deterministic Büchi automata.*

5 Büchi-Definable Objectives

The characterisation of Corollary 19 gives a procedure to compute pseudo-Nash equilibria in games with objectives defined by deterministic Büchi automata (one automaton per player). The algorithm runs in time exponential in the number of players since we have to build the product with all the deterministic Büchi automata defining the objective of a player. We prove that our problems are EXPTIME-hard by encoding two-player countdown games [14] into multiplayer games. Each bit of the countdown will be managed by a different player, who is in charge of checking that this bit is correctly updated at each transition.

▶ **Theorem 21.** *The verification, existence, and constrained existence problems for finite games with objectives defined by deterministic Büchi automata are* EXPTIME*-complete.*

We now prove that when the deterministic Büchi automata defining the objectives are 1-*weak* (i.e. when all strongly connected components of the transition graph consist of just one state), all our three problems can be solved in PSPACE. In particular, this result applies to safety (and reachability) objectives, which can be defined by 1-weak automata. Our algorithm is based on a procedure that, given parameters (P, n, q), computes the set of states s such that in the product game $(s, q) \in \mathrm{Rep}^n(P)$. The procedure computes the repellor as a fixpoint, calling itself recursively on instances (P', n', q'), where either $P' \subsetneq P$, $n' < n$, or q' is a successor of q. The maximal number of nested calls is $|P| + n + \sum_{i \in \mathrm{Agt}} \ell_i$, where ℓ_i is the length of the longest acyclic path in $\mathcal{A}_i$. According to Lemma 8, n can be bounded by a polynomial. The whole computation thus runs in polynomial space.

▶ **Theorem 22.** *The verification, existence, and constrained existence problems are in* PSPACE *for finite games with objectives defined by* 1*-weak deterministic Büchi automata.*

The matching lower bound holds already for the special case of safety objectives.

▶ **Proposition 23.** *The verification, existence, and constrained existence problems are* PSPACE*-hard for finite games with safety objectives.*

Proof. The hardness proof for the verification (and for the constrained existence) problem is by a reduction from QSAT: for every closed quantified Boolean formula ϕ in conjunctive prenex normal form, we construct a game $\mathcal{G}(\phi)$ with initial state s_1 and safety objectives such that $\mathcal{G}(\phi)$ has a pseudo-Nash equilibrium with payoff $(0, \dots, 0)$ from s_1 iff the formula is true. Let $\phi = \exists x_1 \forall x_2 \dots Q_n x_n \; C_1 \wedge C_2 \wedge \dots \wedge C_m$, where each clause C_j is a disjunction of literals over the variables $x_1, \dots, x_n$; we identify C_j with the set of literals occurring in it. Then the game $\mathcal{G}(\phi)$ is played by players $0, 1, \dots, m$. The set of states is $\{s_1, x_1, \neg x_1, \dots, s_n, x_n, \neg x_n, s_{n+1}\}$, and there are transitions from s_i to x_i and $\neg x_i$, and from x_i and $\neg x_i$ to s_{i+1}; additionally, there is a transition from s_{n+1} back to s_{n+1}. If i is odd, then the state s_i is controlled by player 0; otherwise, the game proceeds nondeterministically from s_i to either x_i or $\neg x_i$ (see Figure 3). Player 0 loses every play of the game, i.e. $\mathcal{L}_0 = \emptyset$, whereas for $j > 0$ player j's objective is to avoid the set of literals occurring in the clause C_j, i.e. $\mathcal{L}_j = (\mathrm{States} \setminus C_j)^\omega$. It is easy to see that ϕ is true iff there is a strategy for player 0 such that all outcomes are losing for all players.

To prove hardness of the existence problem, it suffices to add two states s_0 and s_0' to the game $\mathcal{G}(\phi)$: from s_0, the game proceeds nondeterministically to either s_0' or s_1, and we add

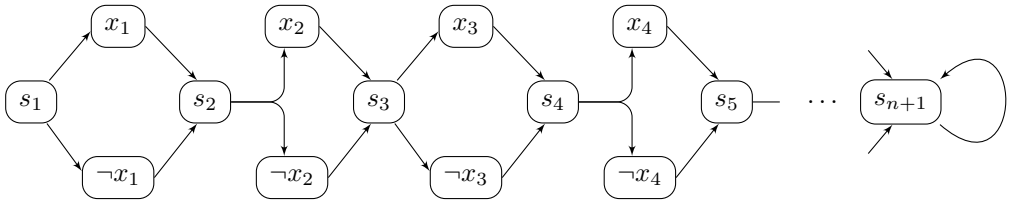


Figure 3 Reducing from QSAT

a transition from s'_0 back to s'_0. Finally, the objective of player 0 is the set $\mathcal{L}_0 = \{s_0, s'_0\}^\omega$. It follows that there is a pseudo-Nash equilibrium from s_0 (with the optimal play leading to s'_0) iff there is a pseudo-Nash equilibrium from s_1 with payoff $(0, \dots, 0)$. ◀

Note that the hardness proof requires nondeterminism. For deterministic games we can solve the problem by guessing the set of losing players and an (ultimately-periodic) path in the corresponding repellor transition system. We then have to check that all possible deviations fall in some repellor set. The best algorithm we could get for this check runs in time $O(|\text{States}|^2 \cdot |\text{Agt}| \cdot |\text{Tab}|^{\log(\max_i |Q_i|)})$, which is only polynomial when the size of the Büchi automata is bounded.

▶ **Theorem 24.** *The verification, existence, and constrained existence problems are in* NP *for finite deterministic games with objectives defined by* 1*-weak deterministic Büchi automata of bounded size.*

The matching lower bound holds again for the special case of safety objectives since no nondeterministic transitions arise in the construction used for proving Proposition 23 when we reduce from SAT (except for the existence problem, where we require a different construction).

▶ **Proposition 25.** *The verification, existence and constrained existence problems are* NP*-hard for finite deterministic games with safety objectives.*

6 Discussion

In this paper we focused on Büchi objectives. The natural next step is to go to parity objectives, which can encode arbitrary ω-regular objectives. In the turn-based case, the constrained existence problem becomes NP-complete for parity (or even co-Büchi) objectives [19]. In fact, we can get the same upper bound in deterministic concurrent games under the assumption that strategies *can* observe actions.

▶ **Theorem 26.** *The constrained existence problem is in* NP *for finite deterministic concurrent games with parity objectives if we assume that strategies can observe actions.*

In Section 2, we mentioned that making actions unobservable by players is a relevant modelling assumption, but that it makes the computation of equilibria harder. This claim is justified by the following result, which is obtained by a reduction from the strategy problem for *generalised parity games* [7].

▶ **Proposition 27.** *The verification, existence and constrained existence problems are* coNP*-hard for finite deterministic concurrent games with parity objectives. In particular, unless* NP = coNP*, these problems do not belong to* NP*.*

A natural question is whether the repellor techniques that we develop can be used to handle imperfect information in a more general sense than just state-based vs. action-based strategies. This is one of our directions for future work.

References

1 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

2 E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In SSC'98, p. 469–474. Elsevier Science, 1998.

3 P. Bouyer, R. Brenguier, and N. Markey. Nash equilibria for reachability objectives in multi-player timed games. In CONCUR'10, LNCS 6269, p. 192–206. Springer, 2010.

4 P. Bouyer, R. Brenguier, and N. Markey. Nash equilibria for reachability objectives in multiplayer timed games. Research Report LSV-10-12, ENS Cachan, France, 2010.

5 K. Chatterjee. Two-player nonzero-sum ω-regular games. In CONCUR'05, LNCS 3653, p. 413–427. Springer, 2005.

6 K. Chatterjee, T. A. Henzinger, and M. Jurdziński. Games with secure equilibria. In LICS'04, p. 160–169. IEEE Comp. Soc. Press, 2004.

7 K. Chatterjee, T. A. Henzinger, and N. Piterman. Generalized parity games. In FoSSaCS'07, LNCS 4423, p. 153–167. Springer, 2007.

8 K. Chatterjee, R. Majumdar, and M. Jurdziński. On Nash equilibria in stochastic games. In CSL'04, LNCS 3210, p. 26–40. Springer, 2004.

9 L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In CONCUR'03, LNCS 2761, p. 142–156. Springer, 2003.

10 M. Félegyházi, J.-P. Hubaux, and L. Buttyán. Nash equilibria of packet forwarding strategies in wireless ad hoc networks. *IEEE Trans. Mobile Comput.*, 5(5):463–476, 2006.

11 A. M. Fink. Equilibrium in a stochastic n-person game. *J. Science in Hiroshima University*, 28(1):89–93, 1964.

12 D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In TACAS'10, LNCS 6015, p. 190–204. Springer, 2010.

13 T. A. Henzinger. Games in system design and verification. In TARK'05, p. 1–4, 2005.

14 M. Jurdziński, F. Laroussinie, and J. Sproston. Model checking probabilistic timed automata with one or two clocks. In TACAS'07, LNCS 4424, p. 170–184. Springer, 2007.

15 J. F. Nash, Jr. Equilibrium points in n-person games. *Proc. National Academy of Sciences of the USA*, 36(1):48–49, 1950.

16 R. Selten. Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 121:301–324 and 667–689, 1965.

17 L. S. Shapley. Stochastic games. *Proc. National Academy of Sciences of the USA*, 39:1095–1100, 1953.

18 W. Thomas. Infinite games and verification (extended abstract of a tutorial). In CAV'02, LNCS 2404, p. 58–64. Springer, 2002.

19 M. Ummels. The complexity of Nash equilibria in infinite multiplayer games. In FoSSaCS'08, LNCS 4962, p. 20–34. Springer, 2008.

20 M. Ummels and D. Wojtczak. The complexity of Nash equilibria in limit-average games. In CONCUR'11, LNCS 6901, p. 482–496. Springer, 2011.

21 M. Ummels and D. Wojtczak. The complexity of Nash equilibria in stochastic multiplayer games. *Logical Methods in Computer Science*, 7(3), 2011.

22 N. Vielle. Two-player stochastic games I–II. *Israel J. of Mathematics*, 119(1):55–126, 2000.

A Perfect-Information Construction for Coordination in Games*

Dietmar Berwanger[1], Łukasz Kaiser[2], and Bernd Puchala[3]

1 **LSV, CNRS &, ENS Cachan, France**
2 **LIAFA, CNRS & Université Paris Diderot – Paris 7, France**
3 **Mathematische Grundlagen der Informatik, RWTH Aachen University, Germany**

Abstract

We present a general construction for eliminating imperfect information from games with several players who coordinate against nature, and to transform them into two-player games with perfect information while preserving winning strategy profiles. The construction yields an infinite game tree with epistemic models associated to nodes. To obtain a more succinct representation, we define an abstraction based on homomorphic equivalence, which we prove to be sound for games with observable winning conditions. The abstraction generates finite game graphs in several relevant cases, and leads to a new semi-decision procedure for multi-player games with imperfect information.

1998 ACM Subject Classification F.1.2. Alternation and nondeterminism

Keywords and phrases Games, Imperfect Information, Epistemic Models, Distributed Synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.387

1 Introduction

A game with perfect information is one where a player knows the state of the play at any stage. If he does not, we speak of a game with imperfect information. Analysing games with perfect information appears conceptually easier than those of imperfect information, which require handling the uncertainty of players. We present a generic construction for eliminating imperfect information from games where players coordinate against nature, and transform them into games with perfect information while preserving winning strategies.

We consider infinite games played on finite graphs [10, 16]. Plays proceed in stages in which a token is moved along the edges, forming an infinite path. A state corresponds to the node of the graph holding the token. Under perfect information, the current state is explicitly announced to each player at every stage. Under imperfect information, the announcement is made with uncertainty modelled by an indistinguishability relation between states.

In our setting, there are n players that form a coalition against nature; at each stage, the players choose simultaneously an action and nature moves the token along an edge compatible with these choices. The objective of the players is to ensure that the outcoming path satisfies a given winning condition, regardless of the moves of nature. We focus on the *coordinated winning strategy problem*: to decide whether the grand coalition has a joint strategy to ensure a win, and to construct one, if this is the case. When we speak about the solution of a game throughout the paper, we mean the solution to both the decision and

* Work supported by the ESF EUROCORES programme LogiCCC and the ESF GAMES grant No. 4505.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 387–398

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the construction variant of the coordinated winning strategy problem. These problems are central to the area of distributed controller synthesis (see [8, 11, 5]).

For the case of a single player against nature, the winning strategy problem has been formulated and solved by Reif 25 years ago [15] – this basic case does not raise the issue of coordination. Reif's approach proceeds by elimination of imperfect information, as the author phrases it: for a given game G with imperfect information, a game G^+ with perfect information is constructed in a way that resembles the powerset construction for determinising finite automata. The states of the perfect-information game G^+ correspond to subsets of states in the imperfect-information game G. Intuitively, any set Π of plays in G that are indistinguishable for the player corresponds to one play π in G^+, and the subset state reached in G^+ via π consists of the states reachable in G when one of the plays from Π is played. The states of G^+ thus represent enough of the player's knowledge about the current state of a play in G to allow transferring his strategies from G^+ to G in a way that preserves winning.

Reif's subset construction allows to reduce (both the decision and the construction variant of) the winning strategy problem for a game with imperfect information played by a player against nature to the corresponding problem in a two-player game of perfect information over a state space that may be exponentially larger. Although the original procedure addressed only games with simple, reachability winning conditions, it extends easily to general observable ω-regular conditions and the resulting games belong to the class of infinite games on finite graphs that is well understood. They are determined with simple strategies (of bounded memory) and they can be solved algorithmically: it is decidable whether a player has winning strategies, and if so, one can construct one (see, e.g., [6]). Thus, the reduction yields solution procedures for the original games of imperfect information and further insights, e.g., about the memory requirement of winning strategies.

Unfortunately, the classical subset construction is not sound in settings that involve more than one player. In fact, the coordinated winning strategy problem is generally undecidable already for two players against nature [12, 13, 17], which implies not only that the subset construction is inadequate for eliminating imperfect information in games with two or more players, but, moreover, that any procedure that transforms a game with imperfect information over a finite graph into one with perfect information over a possibly larger but still finite graph will fail to preserve the solution to the winning strategy problem in the general case. In [1], Arnold and Walukiewicz give a concrete example of a game with two players against nature where winning strategies depend, at each stage, on the number of previous stages – to keep track of this number, a perfect-information variant of the game would require infinitely many states.

We are interested in constructions that generalise Reif's classical approach in the sense that they transform an n-player game G with imperfect information into a two-player zero-sum game G^+ with perfect information such that

(i) the grand coalition in G has a winning strategy against nature if, and only if, the first player has a winning strategy in G^+;
(ii) winning strategies of the first player in G^+ can be translated uniformly into joint winning strategies of the grand coalition in G and vice versa.

If we think of players as components of a system (each with imperfect information about the global state) that shall follow a joint strategy prescribed by the system designer, such a construction allows to formulate the task of the designer in terms of games between two players: the system designer and nature (or the environment, in the phrasing of the distributed-systems literature). One desirable property of such a construction is that it produces instances of perfect-information games that are finite, for possibly large classes of

input games with imperfect information – even if, as pointed out in the previous paragraph, this cannot work for the general case.

Several approaches to identify computationally manageable classes of games with imperfect information among several players have been proposed during the last decade [8, 9, 3, 7, 14, 4]. As a common pattern, tractability is ensured by restricting the way information flows between players.

In this paper, we take a different approach and propose a sufficient, though undecidable, condition for manageability of games with imperfect information. Our perfect-information construction is based on the unravelling of an imperfect-information game as a tree with epistemic models associated to nodes. Intuitively, an epistemic model is a snapshot of what players know at a stage of the game. The unravelling generates a two-player game of perfect information on an infinite tree.

To obtain a more succinct representation, we perform an abstraction by taking the quotient of the tree under homomorphic equivalence of epistemic models. We prove that this abstraction method is sound for imperfect-information games with *observable* ω-regular winning conditions. Consequently, all games that yield a finite quotient admit computable solutions. In particular, this gives an alternative proof for the decidability of games with hierarchical information and observable regular winning conditions. Our proof provides an elementary solution for these games, whereas previous results rely on the simulation theorem of alternation tree automata by nondeterministic ones.

2 Preliminaries

2.1 Distributed Games

We consider games played by n players, $0, 1, \dots, n-1$, against nature. We refer to a list of elements $x = (x_i)_{i<n}$, one for each player, as a *profile*. The *grand coalition* is the set $\{0, \dots, n-1\}$ of all players; nature is not regarded as a player.

Beforehand, we fix a set A_i of *actions* available to Player i, and we denote by A the set of all action profiles. A *distributed game for n players with imperfect information* is described by a structure $\mathcal{G} = (V, \Delta, (\sim_i)_{i<n}, W)$ where V is a finite set of *positions*, $\Delta \subseteq V \times A \times V$ is a *move* relation, and each $\sim_i$ is an equivalence relation on V called the *indistinguishability* relation of Player i. Finally, W is a subset of V^ω describing the *winning condition*.

A *play* in $\mathcal{G}$ is a sequence of positions $\pi = v_0 v_1 v_2 \dots$ such that, for every stage $l \geq 0$, there exists an action profile a_l such that $(v_l, a_l, v_{l+1}) \in \Delta$. We denote the set of all plays by Π. In general, the winning condition is just a set of plays, $W \subseteq \Pi$. We will often focus on ω-regular sets W. More specifically, we will be interested in observable winning conditions. For a set of colours C, we say that a colouring $\Omega : V \to C$ is *observable* if, whenever $\Omega(v) \neq \Omega(w)$, we have $v \not\sim_i w$, for all players i. An *observable winning condition* is described by a pair (Ω, W_o), consisting of an observable colouring Ω and a set of infinite sequences of colours $W_o \subseteq C^\omega$. Then, the associated winning set is $W = \{v_0 v_1 v_2 \dots \mid \Omega(v_0)\Omega(v_1)\Omega(v_2) \dots \in W_o\}$.

A history is a finite prefix of a play. A *strategy* for Player i is a function $\sigma_i : V^* \to A_i$ such that $\sigma_i(\pi) = \sigma_i(\rho)$ for any two histories $\pi, \rho \in V^*$ with $\pi \sim_i^* \rho$, where $\sim_i^*$ is the extension of $\sim_i$ to sequences. A *joint strategy* for the grand coalition is a profile $\sigma = (\sigma_0, \dots, \sigma_{n-1})$ consisting of one strategy σ_i for every player i. We say that a play $\pi = v_0 v_1 \dots$ is *consistent* with σ, if $(v_l, \sigma(v_0 \dots v_l), v_{l+1}) \in \Delta$, for every stage $l > 0$. In this case, we refer to the histories of π as σ-histories. A joint strategy profile σ is *winning* from a position $v_0 \in V$, if each play from v_0 that is consistent with σ belongs to W. We study the following question: given a game $\mathcal{G}$, does the grand coalition have a winning strategy profile for $\mathcal{G}$?

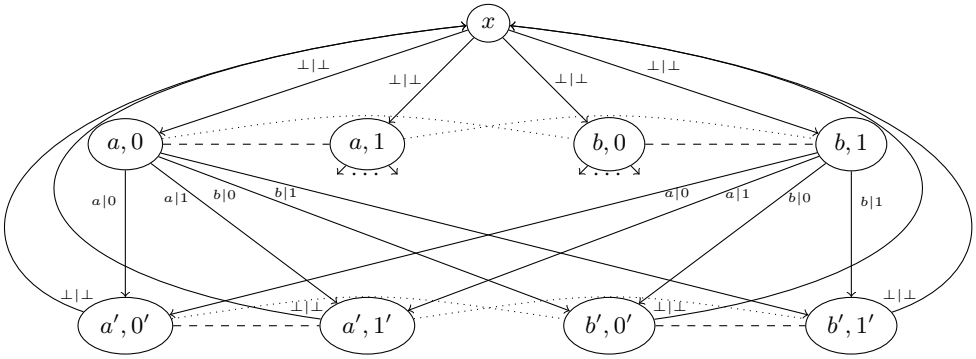


Figure 1 A distributed game $\mathcal{G}_{\|}$.

▶ **Example 1.** Figure 1 describes a distributed game $\mathcal{G}_{\|}$ with two players. The relations $\sim_0$ and $\sim_1$ are represented by dashed and dotted lines, respectively. The game starts at position x where the players have only trivial moves $\bot$ and nature chooses a letter from $\{a, b\}$ and a digit from $\{0, 1\}$. The label of the successor position reflects this choice. Player 0 only observes whether nature has chosen a or b, whereas Player 1 observes whether it was 0 or 1. Next, Player 0 chooses a letter from $\{a, b\}$ and Player 1 a digit from $\{0, 1\}$, again reflected by the label of the successor. After that, the game returns to x for another round.

Let us set $\mathbb{A} = \{a, b\}$, $\mathbb{A}' = \{a', b'\}$, and $\mathbb{D} = \{0, 1\}$, $\mathbb{D}' = \{0', 1'\}$. As one player observes only letters and the other only digits, a strategy f of Player 0 in $\mathcal{G}$ corresponds to a function $(\mathbb{A}\mathbb{A}')^*\mathbb{A} \to \mathbb{A}$, whereas a strategy g of Player 1 corresponds to a function $(\mathbb{D}\mathbb{D}')^*\mathbb{D} \to \mathbb{D}$. Let W be a winning condition in $\mathcal{G}$, i.e. a subset of $(x(\mathbb{A} \times \mathbb{D})(\mathbb{A}' \times \mathbb{D}'))^*$. Then, the strategy profile (f, g) is winning if

$$x\binom{l_1}{d_1}\binom{f(l_1)'}{g(d_1)'}\, x\binom{l_2}{d_2}\binom{f(l_1\, f(l_1)'\, l_2)}{g(d_1\, g(d_1)'\, d_2)}\, x \cdots \in W.$$

Notice that, between two successive visits to position x, nature chooses a pair of bits and each of the players chooses one bit, with the first bit always revealed only to the first player and the second bit only to the second player. This is essentially the game structure considered in [13], where the authors construct regular winning conditions that require the players to construct the run of a given Turing machine. Deciding whether a winning profile (f, g) for the resulting game exists reduces to deciding whether the machine halts on the empty tape. Accordingly, the joint winning strategy problem is undecidable on this class of games.

2.2 Epistemic Models and Homomorphisms

To describe the knowledge acquired by the players during a play, we use epistemic models. An *epistemic model* over $\mathcal{G}$ is a Kripke structure $\mathcal{K} = (K, (P_v)_{v \in V}, (\sim_i)_{i<n})$ where $(P_v)_{v \in V}$ is a partition of K and each $\sim_i$ is an equivalence relation on K such that, for all $k, k' \in K$, if $k \sim_i k'$, then $v_k \sim_i v_{k'}$, with v_k denoting the unique element from V such that $k \in P_{v_k}$. Usually, $\mathcal{K}$ will be connected by $\sim_\cup = \bigcup_i \sim_i$, except when indicated otherwise. Notice that $\sim_\cup$ may not necessarily be an equivalence relation.

We recall the notion of graph homomorphism, which we apply to epistemic models. Let $\mathcal{K} = (K, (P_v)_{v \in V}, (\sim_i)_{i<n})$ and $\mathcal{K}' = (K', (P'_v)_{v \in V}, (\sim'_i)_{i<n})$ be epistemic models. A function f is a *homomorphism* from $\mathcal{K}$ to $\mathcal{K}'$, if $P_v(k) \implies P'_v(f(k))$ and $k \sim_i k' \implies f(k) \sim'_i f(k')$. The models are *homomorphically equivalent*, $\mathcal{K} \approx \mathcal{K}'$, if there exists a homomorphism from

$\mathcal{K}$ to $\mathcal{K}'$ and one from $\mathcal{K}'$ to $\mathcal{K}$. Notice that $\approx$ is an equivalence relation as the composition of two homomorphisms is again a homomorphism.

For a finite epistemic model $\mathcal{K}$, a *core* is a model $\mathcal{K}' \approx \mathcal{K}$ with the minimal number of elements. One crucial observation, which follows for epistemic models in the same way as the standard argument for graphs, is that the core of a model is unique up to isomorphism.

▶ **Lemma 2.** *Every finite epistemic model has a unique core, up to isomorphism.*

3 Epistemic Unfolding

In more traditional approaches to analysing games on graphs, the unfolding collects histories of the original game. We present a new kind of unfolding that uses Kripke structures to collect the full description of the knowledge that players have at a certain stage of the play. When unfolding a game $\mathcal{G}$, we will keep track of the information available to all players in an epistemic model. Thus, the states of the unfolding are epistemic models over $\mathcal{G}$. At the start, we assume that all players know that they are at the initial position, thus the initial epistemic model will be a trivial, one-element structure consisting of $\{v_0\}$, $\mathcal{K}_0 = (\{v_0\}, (P_v)_{v\in V}, (\sim_i)_{i<n})$, where $P_{v_0} = \{v_0\}$, $P_w = \emptyset$ for $w \neq v_0$, and each $\sim_i = \{(v_0, v_0)\}$.

Assume that, in a state of the unfolding represented by an epistemic model $\mathcal{K}$, the players agreed on take actions described by a profile a. What will the epistemic state of a player be after executing these actions? Let $(a_k)_{k\in K}$ be a tuple of action profiles $a_k \in A$ compatible with the players' knowledge, i.e. for every $i < n$ and for all $k, k' \in K$ with $k \sim_i k'$, we have $(a_k)_i = (a_{k'})_i$. We define the, possibly disconnected, epistemic model

$$\mathrm{Update}(\mathcal{K}, (a_k)_{k\in K}) := (K', (P_v)_{v\in V}, (\sim_i)_{i<n}), \text{ by setting}$$

- $K' = \{kv \mid k \in K, k \in P_w \text{ and } (w, a_k, v) \in \Delta\}$,
- $P_v = \{kv \mid kv \in K'\}$,
- $kv \sim_i k'v' \iff k \sim_i^{\mathcal{K}} k'$ and $v \sim_i^{\mathcal{G}} v'$.

The set of epistemic successor models $\mathrm{Next}(\mathcal{K}, (a_k)_{k\in K})$ consists of the $\sim_\cup$-connected components of $\mathrm{Update}(\mathcal{K}, (a_k)_{k\in K})$.

To unfold a game $\mathcal{G}$ and track the knowledge with epistemic models, we start with the initial structure $\mathcal{K}_0$ as above and consider all possible action profiles a the players can take. We get the epistemic models $\mathrm{Next}(\mathcal{K}, a)$ as next states, and continue the unfolding from there. With this dynamic process in mind, we give the following declarative definition.

▶ **Definition 3** (Epistemic Unfolding)**.** The *epistemic unfolding* of a distributed game $\mathcal{G}$ is a game

$$\mathrm{Tr}(\mathcal{G}) := (V^t, \Delta^t, (\sim_i)_{i<n}, W^t), \text{ where}$$

- V^t is the set of all epistemic models $\mathcal{K}$ over $\mathcal{G}$ with $K \subseteq V^*$,
- $\Delta^t = \{(\mathcal{K}, (a_k)_{k\in K}, \mathcal{K}') \mid (a_k)_{k\in K} \in A^{|K|} \text{ and } \mathcal{K}' \in \mathrm{Next}(\mathcal{K}, (a_k)_{k\in K})\}$,
- $\sim_i = \{(\mathcal{K}, \mathcal{K}) \mid \mathcal{K} \in V^t\}$, i.e. $\mathrm{Tr}(\mathcal{G})$ is a game with perfect information,
- $\mathcal{K}_0\mathcal{K}_1 \cdots \in W^t$ if, and only if, for *each* sequence $\pi = k_0k_1 \ldots$ such that $k_l \in \mathcal{K}_l$ and $k_{l+1} = k_lv$ for some v, with $(v_{k_l}, a, v) \in \Delta$ for some a, it holds that $v_{k_0}v_{k_1} \cdots \in W$.

Note that the actions in the game $\mathrm{Tr}(\mathcal{G})$ correspond to tuples of actions in the original game: At a position $\mathcal{K}$, each player i chooses a tuple of actions $((a_k)_i)_{k\in K}$, one for every world of the epistemic model $\mathcal{K}$. The tuple of action profiles $(a_k)_{k\in K}$ yields the set $\mathrm{Next}(\mathcal{K}, (a_k)_{k\in K})$ of successor models from which nature chooses the next position.

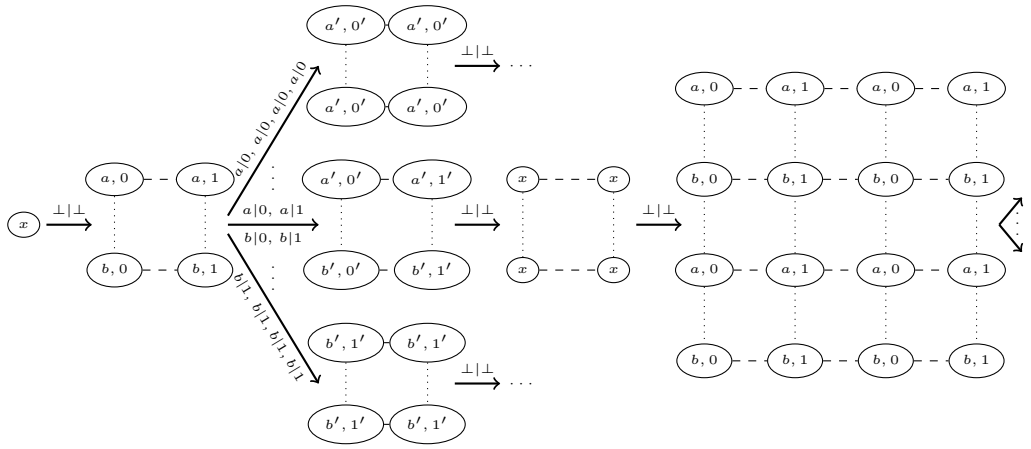


Figure 2 Epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{\|})$ of the game $\mathcal{G}_{\|}$.

The winning condition of $\mathrm{Tr}(\mathcal{G})$ requires that all paths through the sequence of Kripke structures be winning in the original game. Let us detail this for the case of observable winning conditions (Ω, W_o). Since epistemic models are $\sim_\cup$-connected, the colouring Ω is constant for all worlds of a position $\mathcal{K} \in \mathrm{Tr}(\mathcal{G})$; we write $\Omega(\mathcal{K})$ for this colour. Then, we have $\mathcal{K}_0\mathcal{K}_1 \cdots \in W^t$ if, and only if, $\Omega(\mathcal{K}_1)\Omega(\mathcal{K}_2) \ldots \in W_o$. Notice however, that this description is not valid for infinite game graphs, as there may be infinite plays in $\mathrm{Tr}(\mathcal{G})$ for which there is no corresponding infinite play in $\mathcal{G}$. In the case of finite game graphs the above remark follows by König's Lemma.

Observe that, since $\mathrm{Tr}(\mathcal{G})$ is a game with perfect information, in particular all players of the grand coalition have the same information. Thus, the grand coalition can be regarded as a single super-player who chooses actions on behalf of every member of the coalition, and the game can be solved as if it was a two-player game between this super-player and nature (now regarded as a second player).

▶ **Example 4.** In Figure 2, we represent a few first steps of the epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{\|})$ of the game $\mathcal{G}_{\|}$ from Example 1. Note that the structures get larger as more and more knowledge of the players has to be accounted for. Also observe that, in contrast to the standard unfolding, the branching factor in $\mathrm{Tr}(\mathcal{G}_{\|})$ may grow with increasing level.

The following theorem explains the basic utility of the epistemic unfolding.

▶ **Theorem 5.** *The grand coalition has a winning strategy in the distributed game $\mathcal{G}$ from v_0 if, and only if, the grand coalition has a winning strategy in* $\mathrm{Tr}(\mathcal{G})$ *from $\mathcal{K}_0$.*

Proof. ($\Rightarrow$) First, let $\sigma = (\sigma_0, \ldots, \sigma_{n-1})$ be a winning strategy for the coalition in $\mathcal{G}$ from v_0. We define the strategy $\sigma^t = (\sigma_0^t, \ldots, \sigma_{n-1}^t)$ for the coalition for $\mathrm{Tr}(\mathcal{G})$ by induction over the length of histories of $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$ such that, for each history $\pi = \mathcal{K}_0 \ldots \mathcal{K}_r$ consistent with σ^t, every $\pi \in K_r$ is consistent with σ. Note that, in each step r, we only need to extend σ^t to histories of length $r+1$ that are consistent with σ^t. For $r = 0$ the statement is trivial. Let now $\pi^t = \mathcal{K}_0 \ldots \mathcal{K}_r$ be an arbitrary history of $\mathrm{Tr}(\mathcal{G})$ that is consistent with σ^t. We define $\sigma^t(\pi^t) = (a_k)_{k \in K_r}$ by setting $a_k = \sigma(k)$, for every $k \in K$. Notice that each $k \in K_r$ is a σ-history of $\mathcal{G}$ from v_0. We observe that $(a_k)_{k \in K_r} \in \mathrm{act}(\mathcal{K}_r)$: If $k \sim_i k'$, then $k \sim_i^* k'$, and since σ_i is a strategy for player i for $\mathcal{G}$, we have $(a_k)_i = \sigma_i(k) = \sigma_i(k') = (a_{k'})_i$.

Now consider a model $\mathcal{K}_{r+1} \in \mathrm{Next}(\mathcal{K}_r, (a_k)_{k\in K_r})$. By definition, $\pi^t\mathcal{K}_{r+1}$ is consistent with σ^t and every $\pi \in K_{r+1}$ is consistent with σ. This concludes the induction argument.

Next, consider any play $\pi^t = \mathcal{K}_0\mathcal{K}_1 \dots$ in $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$ consistent with σ^t. Let now $\rho = k_0k_1 \dots$ be any path through the structures in π^t. Since $k_0 = v_0$ and, by construction, each $k_i = v_0 \dots v_i$ such that $v_0 \dots v_i$ is a history consistent with σ, we get that $v_0v_1 \dots \in W$, and thus $\pi^t \in W^t$, by definition. Hence, σ^t is a winning strategy.

($\Leftarrow$) Now let $\sigma^t = (\sigma^t_0, \dots, \sigma^t_{n-1})$ be a winning strategy for the coalition in $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$. We define the strategy $\sigma = (\sigma_0, \dots, \sigma_{n-1})$ for the coalition for $\mathcal{G}$ by induction over the length of histories of $\mathcal{G}$ from $\mathcal{K}_0$ and, simultaneously, with each σ-history $\pi = v_0 \dots v_r$ of $\mathcal{G}$, we associate a history $\zeta(\pi) = \mathcal{K}_0 \dots \mathcal{K}_r$ of $\mathrm{Tr}(\mathcal{G})$ from v_0, such that the following holds.

(i) $\pi \in K_r$;
(ii) if $\rho \sim_i^* \pi$ for some σ-history ρ in $\mathcal{G}$ from v_0 and some $i < n$, then $\zeta(\rho) = \zeta(\pi)$;
(iii) $\zeta(\pi)$ is consistent with σ^t;
(iv) $\zeta(v_0 \dots v_l) = \mathcal{K}_0 \dots \mathcal{K}_l$ for any $l \leq r$.

Note that in each step r, we only need to extend σ to histories of length $r+1$ that are consistent with σ. For $\pi = v_0$ we take $\mathcal{K}_0$ as defined before. Now let $\pi = v_0 \dots v_r$ be any history of $\mathcal{G}$ from v_0 that is consistent with σ and let $\zeta(\pi) = \mathcal{K}_0 \dots \mathcal{K}_r$. We define $\sigma_i(\pi) = (a_\pi)_i$, where $(a_k)_{k\in K_r} := \sigma^t(\zeta(\pi))$, that means, $\sigma_i(\pi)$ is the projection to the i-th component of the action, chosen by player i at $\zeta(\pi)$ for the position $\pi \in K_r$ according to σ^t. First, we observe that σ_i is constant over $\sim_i^*$-equivalence classes: if $\rho \sim_i^* \pi$ for some σ-history ρ of $\mathcal{G}$ from v_0, then by condition (i) and (ii) we have $\rho \in \zeta(\rho) = \zeta(\pi)$, so $\sigma_i(\rho) = (a_\rho)_i$. Moreover, as $\pi \sim_i^* \rho$ and $(a_k)_{k\in K_r} \in \mathrm{act}(\mathcal{K}_r)$, $(a_\pi)_i = (a_\rho)_i$.

Now let $v_{r+1} \in V$ such that $(v_r, \sigma(\pi), v_{r+1}) \in \Delta$ (i.e., πv_{r+1} is a σ-history) and let $\mathcal{K}_{r+1} \in \mathrm{Next}(\mathcal{K}_r, (a_k)_{k\in K_r})$ such that $\pi v_{r+1} \in K_{r+1}$, that means, $\mathcal{K}_{r+1}$ is the unique $\sim_\cup$-connected component of the epistemic model $\mathrm{Update}(\mathcal{K}, (a_k)_{k\in K_r})$ that contains πv_{r+1}. Observe that, since $\mathrm{last}(\pi) = v_r$ and $(v_r, a_\sigma, v_{r+1}) \in \Delta$, the history πv_{r+1} is contained in $\mathrm{Update}(\mathcal{K}, (a_k)_{k\in K_r})$, ensuring (i) and by induction (iv). By definition, $\zeta(\pi v_{r+1}) = \zeta(\pi)\mathcal{K}_{r+1}$ is consistent with σ^t, ensuring (iii), so it remains to show (ii), i.e. that if $\rho v \sim_i^* \pi v_{r+1}$ for some σ-history ρv of $\mathcal{G}$ from v_0 and some $i < n$, then $\zeta(\rho v) = \zeta(\pi v_{r+1})$.

First, notice that $\rho v \sim_i^* \pi v_{r+1}$ implies $\rho \sim_i^* \pi$, so $\zeta(\rho) = \zeta(\pi)$. Moreover, the construction of $\zeta(\pi v_{r+1})$ from $\zeta(\pi) = \zeta(\rho)$ is independent of πv_{r+1}, except for the choice of the $\sim_\cup$-connected component $\mathcal{K}_{r+1} \in \mathrm{Next}(\mathcal{K}_r, a)$ of $\mathrm{Update}(\mathcal{K}, (a_k)_{k\in K_r})$. As ρv is a σ-history with $\rho \in \mathcal{K}_r$, by definition of $\sigma(\rho)$, we have $\rho v \in \mathrm{Update}(\mathcal{K}_r, (a_k)_{k\in K_r})$, since $\rho v \sim_i^* \pi v_{r+1}$, ρv and πv_{r+1} lie in the same $\sim_\cup$-connected component of $\mathrm{Update}(\mathcal{K}_r, (a_k)_{k\in K_r},)$.

Finally, consider any play $\pi = v_0v_1 \dots$ in $\mathcal{G}$ from v_0 that is consistent with σ and let $\pi^t = \mathcal{K}_0\mathcal{K}_1 \dots$ be the play in $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$ associated with π, i.e. $\zeta(v_0 \dots v_l) = \mathcal{K}_0 \dots \mathcal{K}_l$ for all l. By construction, any finite prefix $v_0v_1 \dots v_l$ is also a path through π^t of the form $k_0k_1 \dots k_l$, and this extends to the whole play π. Since π^t is consistent with σ^t and thus won by the coalition, by definition of the winning condition W^t, we get that $\pi \in W$. ◀

4 Epistemic Unfolding up to Homomorphic Equivalence

We turn to the task of representing the game $\mathrm{Tr}(\mathcal{G})$ more succinctly. One simple approach would be to identify isomorphic epistemic models; then, strategies can be transferred by isomorphism. To obtain a more significant degree of succinctness, we show that, if the winning condition is observable, it is sufficient to distinguish epistemic models up to homomorphic

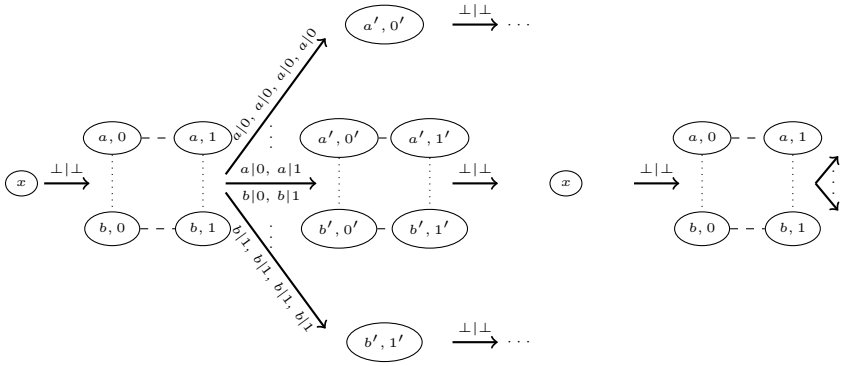


Figure 3 Epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{\|})$ quotiented by core.

equivalence. Consequently, we may take the core of each model (or any retract) instead of the model itself while unfolding.

Essentially, epistemic unfolding up to homomorphism consists of performing the tracking construction while identifying homomorphically equivalent models. Since there may be many possible models equivalent to a model $\mathcal{K}$, we describe this unfolding with respect to a function q, defined on all epistemic models, which chooses for every model $\mathcal{K}$ a homomorphically equivalent companion model $q(\mathcal{K}) \approx \mathcal{K}$.

Although unfolding up to homomorphism is sound only for observable winning conditions (Ω, W_o), we first define the notion for arbitrary winning conditions W. As in the case of the tracking $\mathrm{Tr}(\mathcal{G})$ for games with observable winning conditions, the following definition can be phrased equivalently using sets of colour sequences $W_o \in C^\omega$ to describe winning conditions.

▶ **Definition 6** (Epistemic Unfolding up to Homomorphic Equivalence)**.**
The epistemic unfolding of a distributed $\mathcal{G}$ *up to homomorphic equivalence*, with respect to a function q, is a game

$$\mathrm{Tr}^q(\mathcal{G}) := (V^q, \Delta^q, (\sim_i)_{i<n}, W^q), \text{ where}$$

- V^q is the set $\{q(\mathcal{K}) \mid \mathcal{K} \text{ is an epistemic model over } \mathcal{G}\}$,
- $\Delta^q = \{(\mathcal{K}, (a_k)_{k\in K}, q(\mathcal{K}')) \mid (a_k)_{k\in K} \in A^{|K|} \text{ and } \mathcal{K}' \in \mathrm{Next}(\mathcal{K}, (a_k)_{k\in K})\}$,
- $\sim_i = \{(\mathcal{K}, \mathcal{K}) \mid \mathcal{K} \in V^q\}$, i.e. $\mathrm{Tr}^q(\mathcal{G})$ is a game with perfect information,
- $\mathcal{K}_0\mathcal{K}_1 \cdots \in W^q$ if, and only if, for *each* sequence $\pi = k_0k_1 \ldots$ such that $k_l \in \mathcal{K}_l$ and $k_{l+1} = q(k_l v)$ for some v, with $(v_{k_l}, a, v) \in \Delta$ for some a, it holds that $v_{k_0}v_{k_1} \cdots \in W$.

▶ **Example 7.** We are particularly interested in the case when the image of the homomorphism is the core, i.e., $q(\mathcal{K}) = \mathrm{core}(\mathcal{G})$. In Figure 2, we presented a few positions from the epistemic unfolding $\mathrm{Tr}(\mathcal{G}_{\|})$. In Figure 3 we present the same situation, but these structures are now replaced by their cores. Note that, for example,  gets quotiented to  and thus, from the fourth stage, the structures are repeated. Since we identify isomorphic Kripke structures, the game $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G}_{\|})$ is a finite game with perfect information.

Note that, since $\mathcal{K} \approx \mathcal{K}$, the unfolding Tr^q is a generalisation of the tracking construction Tr obtained with $q(\mathcal{K}) = \mathcal{K}$. We will extend Theorem 5 to all unfoldings Tr^q for games with observable winning conditions. The key point is how to extend the homomorphisms from a model to the next one in a tracking. This is an interesting observation in itself, we formulate it as a separate lemma.

▶ **Lemma 8.** *Let $\mathcal{K}$ and $\mathcal{L}$ be epistemic models, let $h : \mathcal{K} \to \mathcal{L}$ be a homomorphism, and let $(b_l)_{l\in L}$ be a tuple of actions for $\mathcal{L}$. Then $(a_k)_{k\in K}$ with $a_k = b_{h(k)}$ is a tuple of actions for $\mathcal{K}$, and for each connected component $\mathcal{K}'$ of* $\mathrm{Update}(\mathcal{K}, (a_k)_{k\in K})$*, there is a connected component $\mathcal{L}'$ of* $\mathrm{Update}(\mathcal{L}, (b_l)_{l\in L})$ *such that there is a homomorphism $h' : \mathcal{K}' \to \mathcal{L}'$.*

Proof. Since h is a homomorphism, $(a_k)_{k\in K}$ is obviously a tuple of actions for $\mathcal{K}$. Let $\mathcal{K}'$ be a connected component of $\mathrm{Update}(\mathcal{K}, (a_k)_{k\in K})$ and consider the connected component of $\mathrm{Update}(\mathcal{L}, (b_l)_{l\in L})$ that contains all elements $h(k)v$ with $kv \in K'$. Note that since $\mathcal{K}'$ is connected by $\sim_\cup$ and h is a homomorphism, the elements $h(k)v$ are $\sim_\cup$-connected as well and thus are included in a single $\mathcal{L}'$, which we denote by $h(\mathcal{K}')$. The mapping $h' : \mathcal{K}' \to \mathcal{L}'$ with $h'(kv) = h(k)v$ is again a homomorphism, now from $\mathcal{K}'$ to $\mathcal{L}'$. ◀

▶ **Theorem 9.** *Let $\mathcal{G}$ be a distributed game with observable winning condition (Ω, W_o). Then, for all q, the following are equivalent.*

(1) The grand coalition has a winning strategy for $\mathcal{G}$ from v_0.
(2) The grand coalition has a winning strategy for $\mathrm{Tr}(\mathcal{G})$ from $\mathcal{K}_0$.
(3) The grand coalition has a winning strategy for $\mathrm{Tr}^q(\mathcal{G})$ from $q(\mathcal{K}_0)$.

Proof. The equivalence of (1) and (2) was shown already in Theorem 5.

(2) ⇒ (3) Let σ^t be a joint winning strategy for the grand coalition in $\mathrm{Tr}(\mathcal{G})$. We define the joint winning strategy σ^q for the grand coalition in $\mathrm{Tr}^q(\mathcal{G})$ by induction on the length of histories in $\mathrm{Tr}^q(\mathcal{G})$ and simultaneously, with each such history $\pi^q = \mathcal{L}_0\mathcal{L}_1 \dots \mathcal{L}_r$ that is consistent with σ^q, we associate a history $\mu(\pi^q) = \mathcal{K}_0\mathcal{K}_1 \dots \mathcal{K}_r$ in $\mathrm{Tr}(\mathcal{G})$, such that the following conditions hold:

(i) $\mu(\pi^q)$ is consistent with σ^t;
(ii) there is a homomorphism $\nu : \mathcal{L}_r \to \mathcal{K}_r$;
(iii) $\mu(\mathcal{L}_0\mathcal{L}_1 \dots \mathcal{L}_s) = \mathcal{K}_0\mathcal{K}_1 \dots \mathcal{K}_s$ for each $s \leq r$.

For $r = 1$, there is only one history $\pi^q = \mathcal{L}_0 = q(\mathcal{K}_0)$, thus $\mu(\pi^q) = \mathcal{K}_0$ and the homomorphism $\nu : \mathcal{L}_0 \to \mathcal{K}_0$ is obtained from $\mathcal{K}_0 \approx q(\mathcal{K}_0)$. In the following, for an epistemic model $\mathcal{K}$, let $\varphi_\mathcal{K}$ always denote a homomorphism $\varphi : q(\mathcal{K}) \to \mathcal{K}$; in this notation we write $\nu = \varphi_{\mathcal{K}_0}$. Let now $\pi^q_r = \mathcal{L}_0 \dots \mathcal{L}_r$ be a history consistent with σ^q, let $\mu(\pi^q_r) = \mathcal{K}_0 \dots \mathcal{K}_r$ be the associated history consistent with σ^t, and let $\nu : \mathcal{L}_r \to \mathcal{K}_r$ be a homomorphism according to (ii). Consider the actions $(a_k)_{k\in K_r} = \sigma^t(\mu(\pi^q_r))$ prescribed by σ^t, given the history $\mu(\pi^q_r)$ in the game $\mathrm{Tr}(\mathcal{G})$. We define $\sigma^q(\pi^q_r)$ by $\sigma^q(\pi^q_r)(l) = a_{\nu(l)} = \sigma^t(\mu(\pi^q_r))(\nu(l))$. By Lemma 8, $\sigma^q(\pi^q_r)$ is a tuple of actions for $\mathcal{L}_r$. So, for any connected component $\mathcal{L}'$ of $\mathrm{Update}(\mathcal{L}_r, \sigma^q(\pi^q_r))$, the sequence $\pi^q_{r+1} = \pi^q_r\mathcal{L}_{r+1}$ with $\mathcal{L}_{r+1} = q(\mathcal{L}')$ is a history of $\mathrm{Tr}^q(\mathcal{G})$ that is, by definition, consistent with σ^q. Moreover, Lemma 8 yields a homomorphism $\eta : \mathcal{L}' \to \mathcal{K}'$ for a connected component $\mathcal{K}'$ of $\mathrm{Update}(\mathcal{K}_r, \sigma^t(\mu(\pi^q)))$. So, by composing the homomorphism $\varphi_{\mathcal{L}_{r+1}}$ from $\mathcal{L}_{r+1}$ to $q(\mathcal{L}_{r+1}) = \mathcal{L}'$ with the homomorphism η from $\mathcal{L}'$ to $\mathcal{K}'$ we obtain a homomorphism $\nu' : \mathcal{L}_{r+1} \to \mathcal{K}'$ and we set $\mu(\pi^q_{r+1}) = \mu(\pi^q_r)\mathcal{K}'$. By construction, $\mu(\pi^q_{r+1})$ is consistent with σ^t.

Now let $\pi^q = \mathcal{L}_0\mathcal{L}_1 \dots$ be a play in $\mathrm{Tr}^q(\mathcal{G})$ consistent with σ^q. By (iii), the sequence $\mu(\mathcal{L}_0), \mu(\mathcal{L}_0\mathcal{L}_1), \dots$ yields a play $\mathcal{K}_0\mathcal{K}_1 \dots$ in $\mathrm{Tr}(\mathcal{G})$ consistent with σ^t such that, for each $r \in \mathbb{N}$, there is a homomorphism $\nu_r : \mathcal{L}_r \to \mathcal{K}_r$. To show that π^q is indeed a winning play, consider any sequence $\pi = l_0l_1 \dots$ with $l_r \in L_r$ and $l_{r+1} = q(l_rv)$ for v with $(v_{l_r}, a, v) \in \Delta$ for some a. In particular, $\nu_r(l_r) \in K_r$ for each $r \in \mathbb{N}$, so $\Omega(v_{l_0})\Omega(v_{l_1}) \dots = \Omega(\mathcal{K}_0)\Omega(\mathcal{K}_1) \dots$ and as $\mathcal{K}_0\mathcal{K}_1 \dots \in W^q$ we have $\Omega(v_{l_0})\Omega(v_{l_1}) \dots \in W_o$, which proves that π^q is winning. Hence, σ^q is a winning strategy for the grand coalition.

(3) ⇒ (2) This direction follows analogously, by symmetry of homomorphic equivalence. ◀

Notice that, despite the symmetric argument in the proof, if q maps an epistemic model to its core, then $\varphi : \mathcal{K} \to q(\mathcal{K})$ is surjective while $\varphi : q(\mathcal{K}) \to \mathcal{K}$ is injective. This allows to prove the implication from (3) to (2) even in the case of winning conditions that are not observable. However, the implication from (2) to (3) does not hold in general.

While the theorem above can be applied to an arbitrary tracking Tr^q such that $q(\mathcal{K}) \approx \mathcal{K}$, we will concentrate on a specific one, namely $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$, obtained with the function that maps every structure $\mathcal{K}$ to its core. The uniqueness of a core allows us to prove the following remarkable property.

▶ **Theorem 10.** *There exists a finite tracking $\mathrm{Tr}^q(\mathcal{G})$ of $\mathcal{G}$ if, and only if, the tracking $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ is finite.*

As a consequence, we obtain a semi-decision procedure for solving distributed games with observable winning conditions: compute $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ and if it is finite, solve the resulting game with perfect information. The procedure thus takes arbitrary games with observable winning condition as input. This is in contrast with tree-automata based methods, which require a certain information-order among the players, and hence a-priori restrict possible inputs.

5 Hierarchical Games

We present an application of our construction to hierarchical games, which were studied in [18] and are related to the ones in [13, 7]. In particular, they subsume observable ω-regular games with imperfect information where one player has perfect information.

A game $\mathcal{G} = (V, \Delta, (\sim_i)_{i<n}, W)$ is *hierarchical*, if $\sim_0 \subseteq \sim_1 \subseteq \ldots \subseteq \sim_{n-1}$, i.e., if the knowledge of the players is ordered linearly: Player 0 is the best informed one, and Player $n-1$ knows the least. The following theorem provides us with a bound on the size of the game of perfect information obtained by the epistemic unfolding up to homomorphic equivalence of a hierarchical game with imperfect information.

▶ **Theorem 11.** *Let V be a finite set and $n \in \mathbb{N}$. Up to homomorphic equivalence, there are at most $\exp_n(|V|)$ different Kripke structures $\mathcal{K} = (K, (P_v)_{v\in V}, (\sim_i)_{i<n})$ such that:*

1. *$(P_v)_{v\in V}$ is a partition of K*
2. *$\sim_1 \subseteq \ldots \subseteq \sim_n$ are equivalence relations*
3. *$\mathcal{K}$ is connected by $\bigcup_{i=1}^{n} \sim_i$.*

Proof. We denote by $\Psi_n(V)$ the class of all Kripke structures $\mathcal{K} = (K, (P_v)_{v\in V}, (\sim_i)_{i<n})$ with the properties 1. - 3, and we write $\sim_{\cup}^{n} := \bigcup_{i=0}^{n-1} \sim_i$. We prove by induction that, for each $n \in \mathbb{N}$, there is a class $\Psi_n^{\approx}(V)$ of Kripke structures from $\Psi_n(V)$ with $|\Psi_n^{\approx}(V)| = \exp_n(|V|)$ such that each structure from $\Psi_n(V)$ is homomorphically equivalent to one from $\Psi_n^{\approx}(V)$.

First, we define $\Psi_1^{\approx}(V)$ as the set of all Kripke structures $\mathcal{K} = (K, (P_v)_{v\in V}, \sim_0)$ with $K \subseteq V$, $P_v = \{v\}$ for $v \in V$ and $\sim_0 = K \times K$. Hence, any structure in $\Psi_1^{\approx}(V)$ can be identified with a subset of V, $|\Psi_1^{\approx}(V)| = 2^{|V|} = \exp_1(|V|)$. Clearly, $\Psi_1^{\approx}(V) \subseteq \Psi_1(V)$. Let $\mathcal{L} = (L, (P_v)_{v\in V}, \sim_0)$ be any Kripke structure from $\Psi_1(V)$. This structure is connected by $\sim_0 = L \times L$ and we define a homomorphism ν on $\mathcal{L}$ by $\nu(l) = v$, for the unique $v \in V$ such that $l \in P_v$. The homomorphic image $\nu(\mathcal{L}) = (K, (P_v)_{v\in V}, \sim_0)$ of ν is in $\Psi_1^{\approx}(V)$ and $\eta : \nu(\mathcal{L}) \to \mathcal{L}$ with $\eta(v) = l$ for some $l \in L \cap P_v$ is a homomorphism on $\nu(\mathcal{L})$. Hence, $\mathcal{L}$ and $\nu(\mathcal{L})$ are homomorphically equivalent.

For $n > 1$, suppose $\Psi_{n-1}^{\approx}(V)$ has already been constructed. Without loss, we assume that all Kripke structures from $\Psi_{n-1}^{\approx}(V)$ are pairwise disjoint. We define $\Psi_n^{\approx}(V)$ as the set of all Kripke structures $\mathcal{K} = (K, (P_v)_{v\in V}, (\sim_i)_{i<n})$ that consist of a union of epistemic

models from $\Psi^{\approx}_{n-1}(V)$ and we set $\sim_{n-1} = K \times K$. Hence, any structure in $\Psi^{\approx}_{n}(V)$ can be identified with a subset of $\Psi^{\approx}_{n-1}(V)$, so $|\Psi^{\approx}_{n}(V)| = 2^{\exp_{n-1}(|V|)} = \exp_n(|V|)$. Now, let $\mathcal{L} = (L, (P_v)_{v\in V}, (\sim_i)_{i<n})$ be any Kripke structure from $\Psi_n(V)$. As $\mathcal{L}$ is connected by $\sim^n_{\cup}$, we have $\sim_{n-1} = L \times L$: any $l, l' \in L$ are connected in $\mathcal{L}$ via some $\sim^n_{\cup}$-path and as $\sim^{n-1}_{\cup} \subseteq \sim_{n-1}$ and $\sim_{n-1}$ is transitive, it follows that $l \sim_{n-1} l'$.

Consider the decomposition of $\mathcal{L}$ into $\sim^{n-1}_{\cup}$-connected components $\mathcal{L}_1, \ldots, \mathcal{L}_r$. Clearly, $\mathcal{L}_j \in \Psi_{n-1}(V)$ for $j = 1, \ldots, r$ and hence, each $\mathcal{L}_j$ is homomorphically equivalent to a Kripke structure from $\Psi^{\approx}_{n-1}(V)$. For $j \in \{1, \ldots, r\}$ we fix a homomorphism ν_j on $\mathcal{L}_j$ such that the image $\nu_j(\mathcal{L}_j)$ is in $\Psi^{\approx}_{n-1}(V)$ and a homomorphism η_j from $\nu_j(\mathcal{L}_j)$ to $\mathcal{L}_j$. Moreover, we define the homomorphism ν on $\mathcal{L}$ by $\nu|\mathcal{L}_j = \nu_j$ for $j = 1, \ldots, r$. As the components $\mathcal{L}_j$ are pairwise disjoint, this is well defined and it is easy to see that the homomorphic image $\nu(\mathcal{L})$ is in $\Psi^{\approx}_{n}(V)$. Furthermore, we define $\eta : \nu(\mathcal{L}) \to \mathcal{L}$ as follows. For any $\sim^{n-1}_{\cup}$-connected component $\mathcal{M}$ of $\nu(\mathcal{L})$ there exists $j \in \{1, \ldots, r\}$ such that $\nu_j(\mathcal{L}_j) = \mathcal{M}$ and we define $\eta \restriction \mathcal{M} = \eta_j$, for arbitrary j. Now, η is a homomorphism and hence, $\mathcal{L}$ and $\nu(\mathcal{L})$ are homomorphically equivalent. ◀

▶ **Corollary 12.** *For hierarchical games with observable regular winning conditions, the existence of a joint winning strategy for the grand coalition is decidable.*

Proof. Let $\mathcal{G}$ be a hierarchical game with an observable winning condition (Ω, W_o) such that W_o is regular. By Theorem 9, the grand coalition has a joint winning strategy for $\mathcal{G}$ if, and only if, it has a joint winning strategy for $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$, and by Theorem 11, $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ is finite. Moreover, as we observed previously, the winning condition of $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ can be described as $W^{\mathrm{core}} = \{\mathcal{L}_0\mathcal{L}_1 \ldots \in (V^{\mathrm{core}})^\omega \mid \Omega(\mathcal{L}_0)\Omega(\mathcal{L}_1)\ldots \in W_o\}$ so W^{core} is regular as well. Hence, the existence of a joint winning strategy in $\mathcal{G}$ can be decided by solving the game $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ – a finite game with perfect information and a regular winning condition. ◀

6 Outlook

We introduced the epistemic unfolding $\mathrm{Tr}(\mathcal{G})$ of a distributed game $\mathcal{G}$ to capture the knowledge of players; the resulting structure is infinite for all games $\mathcal{G}$ of infinite duration. To obtain a more succinct representation, we restrict to the core of the generated epistemic models and obtain perfect information games $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ that are finite for certain game instances $\mathcal{G}$, in particular for all hierarchical ones. However, we can only guarantee that the quotient $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ preserves winning strategies if the winning condition of $\mathcal{G}$ is observable. Nevertheless, even under observable winning conditions there exist distributed games for which it is undecidable whether a winning strategy profile exists (cf. Propositions 22-24 in [2]).

Theorem 9 and Lemma 8 demonstrate that homomorphic equivalence allows to transfer strategies in observable games. We are persuaded that homomorphic equivalence – and *not* bisimulation – is a suitable notion for a quotient. The current work brought us to the insight that the bisimulation-based tracking introduced in our previous paper [2] does not preserve winning strategies – the assertion of Lemma 14 of the paper is incorrect in the stated form. We are currently preparing an erratum communication on this result, where we will also discuss the appropriateness of homomorphic equivalence in more detail.

The construction of $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ can be done on the fly. Thus, our result provides a semi-decision procedure for the coordinated winning strategy problem for games with imperfect information and observable winning conditions. The procedure halts on all hierarchical games. One important task is to characterise further game classes that are solvable in this way, i.e. instances $\mathcal{G}$ for which $\mathrm{Tr}^{\mathrm{core}}(\mathcal{G})$ is finite. Another challenge is to develop a similar

semi-decision procedure for games with non-observable winning conditions. Does there exists a uniform quotienting function q such that $\mathrm{Tr}^q(\mathcal{G})$ preserves winning strategies for all (even non-observable) winning conditions and is finite for hierarchical games? If not, does such a quotient q_W exists for each regular (non-observable) winning condition W separately?

References

1 André Arnold and Igor Walukiewicz. Nondeterministic controllers of nondeterministic processes. In *Logic and Automata*, volume 2. Amsterdam University Press, 2007.

2 Dietmar Berwanger and Łukasz Kaiser. Information tracking in games on graphs. *Journal of Logic, Language and Information*, 19(4):395–412, 2010.

3 B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. of LICS '05*, pages 321–330. IEEE, 2005.

4 Bernd Finkbeiner and Sven Schewe. Coordination logic. In *Proc. of CSL '10*, volume 6247 of *LNCS*, pages 305–319. Springer, 2010.

5 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games and distributed control for asynchronous systems. In *Proc. of LATIN '04*, volume 2976 of *LNCS*, pages 455–465. Springer, 2004.

6 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

7 Łukasz Kaiser. Game quantification on automatic structures and hierarchical model checking games. In *Proc. of CSL '06*, volume 4207 of *LNCS*, pages 411–425. Springer, 2006.

8 Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *Proc. of LICS '01*, pages 387–396, Washington, DC, USA, 2001. IEEE Computer Society.

9 P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *Proc. of CONCUR '02*, volume 2421 of *LNCS*, pages 145–160, 2002.

10 Robert McNaughton. Infinite Games Played on Finite Graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993.

11 Swarup Mohalik and Igor Walukiewicz. Distributed games. In *Proc. of FSTTCS '03*, volume 2914 of *LNCS*, pages 338–351, 2003.

12 Gary L. Peterson and John H. Reif. Multiple-person alternation. In *Proc. of FOCS '79*, pages 348–363. IEEE, 1979.

13 A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS '90*, pages 746–757. IEEE, 1990.

14 Ramaswamy Ramanujam and Sunil Easaw Simon. A communication based model for games of imperfect information. In *Proc. of CONCUR '10*, volume 6269 of *LNCS*, pages 509–523. Springer, 2010.

15 J. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29:274–301, 1984.

16 Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Proc. of STACS '95*, pages 1–13, 1995.

17 Stavros Tripakis. Undecidable problems of decentralized observation and control on regular languages. *Inf. Process. Lett.*, 90(1):21–28, 2004.

18 R. van der Meyden and T. Wilke. Synthesis of distributed systems from knowledge-based specifications. In *Proc. of CONCUR '05*, pages 562–576. Springer, 2005.

Efficient Approximation of Optimal Control for Continuous-Time Markov Games*

John Fearnley[1], Markus Rabe[2], Sven Schewe[1], and Lijun Zhang[3]

1 **Department of Computer Science, University of Liverpool, Liverpool, United Kingdom**
2 **Department of Computer Science, Universität des Saarlandes, Saarbrücken, Germany**
3 **DTU Informatics, Technical University of Denmark, Lyngby, Denmark**

Abstract

We study the time-bounded reachability problem for continuous-time Markov decision processes (CTMDPs) and games (CTMGs). Existing techniques for this problem use discretisation techniques to break time into discrete intervals of size ε, and optimal control is approximated for each interval separately. Current techniques provide an accuracy of $O(\varepsilon^2)$ on each interval, which leads to an infeasibly large number of intervals. We propose a sequence of approximations that achieve accuracies of $O(\varepsilon^3)$, $O(\varepsilon^4)$, and $O(\varepsilon^5)$, that allow us to drastically reduce the number of intervals that are considered. For CTMDPs, the performance of the resulting algorithms is comparable to the heuristic approach given by Buckholz and Schulz [5], while also being theoretically justified. All of our results generalise to CTMGs, where our results yield the first practically implementable algorithms for this problem. We also provide memoryless strategies for both players that achieve similar error bounds.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.1.2 Approximation

Keywords and phrases Continuous time Markov decision processes and games, Discretisation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.399

1 Introduction

Probabilistic models are being used extensively in the formal analysis of complex systems, including networked, distributed, and most recently, biological systems. Over the past 15 years, probabilistic model checking for discrete-time Markov decision processes (MDPs) and continuous-time Markov chains (CTMCs) has been successfully applied to these rich academic and industrial applications [8, 7, 9, 3]. However, the theory for continuous-time Markov decision processes (CTMDPs), which mix the non-determinism of MDPs with the continuous-time setting of CTMCs [2], is less well developed.

This paper studies the *time-bounded reachability* problem for CTMDPs and their extension to continuous-time Markov games, which is a model with both helpful and hostile non-determinism. This problem is of paramount importance for model checking applications [4]. The non-determinism in the system is resolved by providing a scheduler. The

* This work was partly supported by the Engineering and Physical Science Research Council (EPSRC) through the grant EP/H046623/1 'Synthesis and Verification in Markov Game Structures', the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) of the DFG, and the VKR Centre of Excellence MT-LAB.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 399–410

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

time-bounded reachability problem is to determine or to approximate, for a given set of goal locations G and time bound T, the maximal (or minimal) probability of reaching G before the deadline T that can be achieved by a scheduler.

For CTMCs, this problem can be solved efficiently by the Runge-Kutta method. However, this method requires that the target function can be continuously differentiated four times. Once we move to the CTMDP setting, our target function is not continuously differentiable at all. This is because changing the choice of action at a state introduces a discontinuity in the derivative of the time bounded-reachability probability.

Early work on this problem for CTMDPs focused on restricted classes of schedulers, such schedulers without any access to time in systems with uniform transition rates [1]. Recently however, results have been proved for the more general class of *late schedulers* [13], which will be studied in this paper. The different classes of schedulers are contrasted by Neuhäußer et. al. [12], and they show that late schedulers are the most powerful class. Several algorithms have been given to approximate the time-bounded reachability probabilities for CTMDPs using this scheduler class [4, 6, 13, 15].

The current state-of-the-art techniques for solving this problem are based on different forms of *discretisation*. This technique splits the time bound T into small intervals of length ε. Optimal control is approximated for each interval separately, and these approximations are combined to produce the final result. Current techniques can approximate optimal control on an interval of length ε with an accuracy of $O(\varepsilon^2)$. However, to achieve a precision of π with these techniques, one must choose $\varepsilon \approx \pi/T$, which leads to $O(T^2/\pi)$ many intervals. Since the desired precision is often high (it is common to require that $\pi \leq 10^{-6}$), this leads to an infeasibly large number of intervals that must be considered by the algorithms.

A recent paper of Buckholz and Schulz [5] has addressed this problem for practical applications, by allowing the interval sizes to vary. In addition to computing an approximation of the maximal time-bounded reachability probability, which provides a lower bound on the optimum, they also compute an upper bound. As long as the upper and lower bounds do not diverge too far, the interval can be extended indefinitely. In practical applications, where the optimal choice of action changes infrequently, this idea allows their algorithm to consider far fewer intervals while still maintaining high precision. However, from a theoretical perspective, their algorithm is not particularly satisfying. Their method for extending interval lengths depends on a heuristic, and in the worst case their algorithm may consider $O(T^2/\pi)$ intervals, which is not better than other discretisation based techniques.

Our contribution. In this paper we present a method of obtaining larger interval sizes that satisfies both theoretical and practical concerns. Our approach is to provide more precise approximations for each ε length interval. While current techniques provide an accuracy of $O(\varepsilon^2)$, we propose a sequence of approximations, called double ε-nets, triple ε-nets, and quadruple ε-nets, with accuracies $O(\varepsilon^3)$, $O(\varepsilon^4)$, and $O(\varepsilon^5)$, respectively. Since these approximations are much more precise on each interval, they allow us to consider far fewer intervals while still maintaining high precision. For example, Table 1 gives the number of intervals considered by our algorithms, in the worst case, for a normed CTMDP with time bound $T = 10$.

Of course, in order to become more precise, we must spend additional computational effort. However, the cost of using double ε-nets instead of using current techniques requires only an extra factor of $\log |\Sigma|$, where Σ is the set of actions. Thus, in almost all cases, the large reduction in the number of intervals far outweighs the extra cost of using double ε-nets. Our worst case running times for triple and quadruple ε-nets are not so attractive: triple ε-nets require an extra $|L| \cdot |\Sigma|^2$ factor over double ε-nets, where L is the set of locations,

Table 1 The number of intervals needed by our algorithms for precisions $10^{-7}, 10^{-9}$, and 10^{-11}.

Technique	Error	$\pi = 10^{-7}$	$\pi = 10^{-9}$	$\pi = 10^{-11}$
Current techniques	$O(\varepsilon^2)$	$1,000,000,000$	$100,000,000,000$	$10,000,000,000,000$
Double ε-nets	$O(\varepsilon^3)$	$81,650$	$816,497$	$8,164,966$
Triple ε-nets	$O(\varepsilon^4)$	$3,219$	$14,939$	$69,337$
Quadruple ε-nets	$O(\varepsilon^5)$	605	$1,911$	$6,043$

and quadruple ε-nets require yet another $|L| \cdot |\Sigma|^2$ factor over triple ε-nets. However, these worst case running times only occur when the choice of optimal action changes frequently, and we speculate that the cost of using these algorithms in practice is much lower than our theoretical worst case bounds. Our experimental results with triple ε-nets support this claim.

An added advantage of our techniques is that they can be applied to continuous-time Markov games as well as to CTMDPs. Buckholz and Schulz restrict their analysis to CTMDPs. Moreover, previous works on CTMGs have been restricted to simplified settings, such as the time-abstract setting [4]. Therefore, to the best of our knowledge, we present the first practically implementable approximation algorithms for the time-dependent time-bounded reachability problem in CTMGs. Each of our approximations also provide memoryless strategies for both players that achieve similar error bounds.

2 Preliminaries

▶ **Definition 1.** A continuous-time Markov game (or simply Markov game) is a tuple $(L, L_r, L_s, \Sigma, \mathbf{R}, \mathbf{P}, \nu)$, consisting of a finite set L of locations, which is partitioned into locations L_r (controlled by a *reachability* player) and L_s (controlled by a *safety* player), a finite set Σ of actions, a rate matrix $\mathbf{R} : (L \times \Sigma \times L) \to \mathbb{Q}_{\geqslant 0}$, a discrete transition matrix $\mathbf{P} : (L \times \Sigma \times L) \to \mathbb{Q} \cap [0, 1]$, and an initial distribution $\nu \in \mathit{Dist}(L)$.

We require that the following side-conditions hold: For all locations $l \in L$, there must be an action $a \in \Sigma$ such that $\mathbf{R}(l, a, L) := \sum_{l' \in L} \mathbf{R}(l, a, l') > 0$, which we call *enabled*. We denote the set of enabled actions in l by $\Sigma(l)$. For a location l and actions $a \in \Sigma(l)$, we require for all locations l' that $\mathbf{P}(l, a, l') = \frac{\mathbf{R}(l,a,l')}{\mathbf{R}(l,a,L)}$, and we require $\mathbf{P}(l, a, l') = 0$ for non-enabled actions. We define the *size* $|\mathcal{M}|$ of a Markov game as the number of non-zero rates in the rate matrix $\mathbf{R}$.

A Markov game is called *uniform* with uniformisation rate λ, if $\mathbf{R}(l, a, L) = \lambda$ holds for all locations l and enabled actions $a \in \Sigma(l)$. We further call a Markov game *normed*, if its uniformisation rate is 1. Note that for normed Markov games we have $\mathbf{R} = \mathbf{P}$. We will present our results for normed Markov games only. The following lemma states that our algorithms for normed Markov games can be applied to solve Markov games that are not normed.

▶ **Lemma 2.** *We can adapt an $O(f(\mathcal{M}))$ time algorithm for normed Markov games to solve an arbitrary Markov game in time $O(f(\mathcal{M}) + |L|)$.*

We are particularly interested in Markov games with a single player, which are continuous-time Markov decision processes (CTMDPs). In CTMDPs all positions belong to the reachability player ($L = L_r$), or to the safety player ($L = L_s$), depending on whether we analyse the *maximum* or *minimum* reachability probability problem.

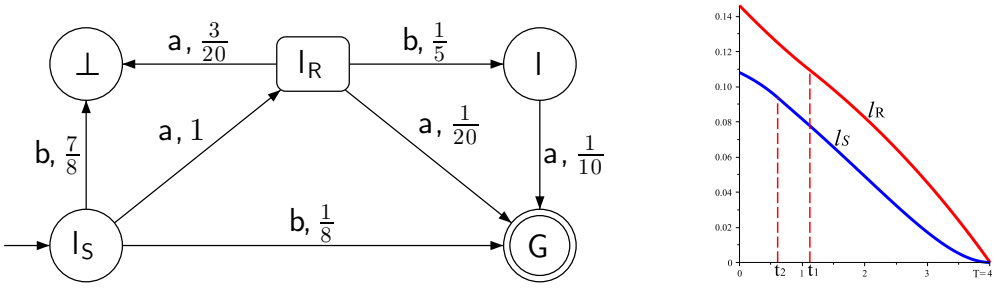


Figure 1 Left: a normed Markov game. Right: the function f within $[0, 4]$ for l_R and l_S.

As a running example, we will use the normed Markov game shown in the left half of Figure 1. Locations belonging to the safety player are drawn as circles, and locations belonging to the reachability player are drawn as rectangles. The self-loops of the normed Markov game are not drawn, but rates assigned to the self loops can be derived from the other rates: for example, we have $\mathbf{R}(l_R, a, l_R) = 0.8$. The locations G and $\bot$ are absorbing, and there is only a single enabled action for l. It therefore does not matter which player owns l, G, and $\bot$.

2.1 Schedulers and Strategies

We consider Markov games in a time interval $[0, T]$ with $T \in \mathbb{R}_{\geq 0}$. The non-determinism in the system needs to be resolved by a pair of strategies for the two players which together form a *scheduler* for the whole system. Formally, a strategy is a function in $Paths_{r/s} \times [0, T] \to \Sigma$, where $Paths_r$ and $Paths_s$ are the sets of finite paths $l_0 \xrightarrow{a_0, t_0} l_1 \ldots \xrightarrow{a_{n-1}, t_{n-1}} l_n$ with $l_n \in L_r$ and $l_n \in L_s$, respectively. We use $\mathcal{S}_r$ and $\mathcal{S}_s$ to denote the strategies of reachability player and the strategies of safety player, respectively, and we use Π_r and Π_s to denote the set of all strategies for the reachability and safety players, respectively. (For technical reasons one has to restrict the schedulers to those which are measurable. This restriction, however, is of no practical relevance. In particular, simple piecewise constant timed-positional strategies $L \times [0, T] \to \Sigma$ suffice for optimal scheduling [14, 13, 2], and all schedulers that occur in this paper are from the particularly tame class of cylindrical schedulers [14].)

If we fix a pair $(\mathcal{S}_r, \mathcal{S}_s)$ of strategies, we obtain a deterministic stochastic process, which is in fact a time inhomogeneous Markov chain, and we denote it by $\mathcal{M}_{\mathcal{S}_{r,s}}$. For $t \leq T$, we use $Pr_{\mathcal{S}_{r+s}}(t)$ to denote the transient distribution at time t over S under the scheduler $(\mathcal{S}_r, \mathcal{S}_s)$.

Given a Markov game $\mathcal{M}$, a goal region $G \subseteq L$, and a time bound $T \in \mathbb{R}_{\geq 0}$, we are interested in the *optimal* probability of being in a goal state at time T (and the corresponding pair of optimal strategies). This is given by:

$$\sup_{\mathcal{S}_r \in \Pi_r} \inf_{\mathcal{S}_s \in \Pi_s} \sum_{l \in G} Pr_{\mathcal{S}_{r+s}}(l, T),$$

where $Pr_{\mathcal{S}_{r+s}}(l, T) := Pr_{\mathcal{S}_{r+s}}(T)(l)$. It is commonly referred to as the *maximum* time-bounded reachability probability problem in the case of CTMDPs with a reachability player only. For $t \leq T$, we define $f : L \times \mathbb{R}_{\geq 0} \to [0, 1]$, to be the optimal probability to be in the goal region at the time bound T, assuming that we start in location l and that t time units have passed already. By definition, it holds then that $f(l, T) = 1$ if $l \in G$ and $f(l, T) = 0$ if $l \notin G$. Optimising the vector of values $f(\cdot, 0)$ then yields the optimal value and its optimal piecewise deterministic strategy.

Let us return to the example shown in Figure 1. The right half of the Figure shows the optimal reachability probabilities, as given by f, for the locations l_R and l_S when the

time bound $T = 4$. The points $t_1 \approx 1.123$ and $t_2 \approx 0.609$ represent the times at which the optimal strategies change their decisions. Before t_1 it is optimal for the reachability player to use action b at l_R, but afterwards the optimal choice is action a. Similarly, the safety player uses action b before t_2, and switches to a afterwards.

2.2 Characterisation of f

We define a matrix $\mathbf{Q}$ such that $\mathbf{Q}(l, a, l') = \mathbf{R}(l, a, l')$ if $l' \neq l$ and $\mathbf{Q}(l, a, l) = -\sum_{l' \neq l} \mathbf{R}(l, a, l')$. The optimal function f can be characterised as a set of differential equations [2], see also [11, 10]. For each $l \in L$ we define $f(l, T) = 1$ if $l \in G$, and 0 if $l \notin G$. Otherwise, for $t < T$, we define:

$$-\dot{f}(l,t) = \operatorname*{opt}_{a \in \Sigma(l)} \sum_{l' \in L} \mathbf{Q}(l, a, l') \cdot f(l', t), \tag{1}$$

where $\mathsf{opt} \in \{\max, \min\}$ is max for reachability player locations and min for safety player locations. We will use the opt-notation throughout this paper.

Using the matrix $\mathbf{R}$, Equation (1) can be rewritten to:

$$-\dot{f}(l,t) = \operatorname*{opt}_{a \in \Sigma(l)} \sum_{l' \in L} \mathbf{R}(l, a, l') \cdot (f(l', t) - f(l, t)) \tag{2}$$

For uniform Markov games, we simply have $\mathbf{Q}(l, a, l) = \mathbf{R}(l, a, l) - \lambda$, with $\lambda = 1$ for normed Markov games. This also provides an intuition for the fact that uniformisation does not alter the reachability probability: the rate $\mathbf{R}(l, a, l)$ does not appear in (1).

3 Approximating Optimal Control for Normed Markov Games

In this section we describe ε-nets, which are a technique for approximating optimal values and strategies in a normed continuous-time Markov game. Thus, throughout the whole section, we fix a normed Markov game $\mathcal{M} = (L, L_r, L_s, \Sigma, \mathbf{R}, \mathbf{P}, \nu)$.

Our approach to approximating optimal control within the Markov game is to break time into intervals of length ε, and to approximate optimal control separately in each of the $\lceil \frac{T}{\varepsilon} \rceil$ distinct intervals. Optimal time-bounded reachability probabilities are then computed iteratively for each interval, starting with the final interval and working backwards in time. The error made by the approximation in each interval is called the *step error*. In Section 3.1 we show that if the step error in each interval is bounded, then the *global error* made by our approximations is also bounded.

Our results begin with a simple approximation that finds the optimal action at the start of each interval, and assumes that this action is optimal for the duration of the interval. We refer to this as the *single* ε-net technique, and we will discuss this approximation in Section 3.2. While it only gives a simple linear function as an approximation, this technique gives error bounds of $O(\varepsilon^2)$, which is comparable to existing techniques.

However, single ε-nets are only a starting point for our results. Our main observation is that, if we have a piecewise polynomial approximation of degree c that achieves an error bound of $O(\varepsilon^k)$, then we can compute a piecewise polynomial approximation of degree $c+1$ that achieves an error bound of $O(\varepsilon^{k+1})$. Thus, starting with single ε-nets, we can construct double ε-nets, triple ε-nets, and quadruple ε-nets, with each of these approximations becoming increasingly more precise. The construction of these approximations will be discussed in Sections 3.3 and 3.4.

In addition to providing an approximation of the time-bounded reachability probabilities, our techniques also provide memoryless strategies for both players. For each level of ε-net, we will define two approximations: the function p_1 is the approximation for the time-bounded reachability probability given by single ε-nets, and the function g_1 gives the reachability probability obtained by following the memoryless strategy that is derived from p_1. This notation generalises to deeper levels of ε-nets: the functions p_2 and g_2 are produced by double ε-nets, and so on.

We will use $\mathcal{E}(k,\varepsilon)$ to denote the difference between p_k and f. In other words, $\mathcal{E}(k,\varepsilon)$ gives the difference between the approximation p_k and the true optimal reachability probabilities. We will use $\mathcal{E}_s(k,\varepsilon)$ to denote the difference between g_k and f. We defer formal definition of these measures to subsequent sections. Our objective in the following subsections is to show that the step errors $\mathcal{E}(k,\varepsilon)$ and $\mathcal{E}_s(k,\varepsilon)$ are in $O(\varepsilon^{k+1})$, with small constants.

3.1 Step Error and Global Error

In subsequent sections we will prove bounds on the *ε-step* error made by our approximations. This is the error that is made in a single interval of length ε. However, in order for our approximations to be valid, they must provide a bound on the *global* error, which is the error made by our approximations over every ε interval. In this section, we prove that, if the ε-step error of an approximation is bounded, then the global error of the approximation is bounded by the sum of these errors.

We define $f : [0,T] \to [0,1]^{|L|}$ as the vector valued function $f(t) \mapsto \bigotimes_{l\in L} f(l,t)$ that maps each point of time to a vector of reachability probabilities, with one entry for each location. Given two such vectors $f(t)$ and $p(t)$, we define the maximum norm $\|f(t)-p(t)\| = \max\{|f(l,t)-p(l,t)| \mid l \in L\}$, which gives the largest difference between $f(l,t)$ and $p(l,t)$.

We also introduce notation that will allow us to define the values at the start of an ε interval. For each interval $[t-\varepsilon,t]$, we define $f_x^t : [t-\varepsilon,t] \to [0,1]^{|L|}$ to be the function obtained from the differential equations (1) when the values at the time t are given by the vector $x \in [0,1]^{|L|}$. More formally, if $\tau = t$ then we define $f_x^t(\tau) = x$, and if $t-\varepsilon \le \tau < t$ and $l \in L$ then we define:

$$-\dot{f}_x^t(l,\tau) = \operatorname*{opt}_{a\in\Sigma(l)} \sum_{l'\in L} \mathbf{Q}(l,a,l') f_x^t(l',\tau). \tag{3}$$

The following lemma states that if the ε-step error is bounded for every interval, then the global error is simply the sum of these errors.

▶ **Lemma 3.** *Let p be an approximation of f that satisfies $\|f(t)-p(t)\| \le \mu$ for some time point $t \in [0,T]$. If $\|f_{p(t)}^t(t-\varepsilon) - p(t-\varepsilon)\| \le \nu$ then we have $\|f(t-\varepsilon)-p(t-\varepsilon)\| \le \mu+\nu$.*

3.2 Single ε-Nets

In single ε-nets, we compute the gradient of the function f at the end of each interval, and we assume that this gradient remains constant throughout the interval. This yields a *linear* approximation function p_1, which achieves a local error of ε^2.

We now define the function p_1. For initialisation, we define $p_1(l,T) = 1$ if $l \in G$ and $p_1(l,T) = 0$ otherwise. Then, if p_1 is defined for the interval $[t,T]$, we will use the following procedure to extend it to the interval $[t-\varepsilon,T]$. We first determine the optimising enabled

actions for each location for $f^t_{p_1(t)}$ at time t. That is, we choose, for all $l \in L$, an action:

$$a^t_l \in \arg\operatorname*{opt}_{a \in \Sigma(l)} \sum_{l' \in L} \mathbf{Q}(l, a, l') \cdot p_1(l', t). \tag{4}$$

We then fix $c^t_l = \sum_{l' \in L} \mathbf{Q}(l, a^t_l, l') \cdot p_1(l', t)$ as the descent of $p_1(l, \cdot)$ in the interval $[t - \varepsilon, t]$. Therefore, for every $\tau \in [0, \varepsilon]$ and every $l \in L$ we have:

$$-\dot{p}_1(l, t - \tau) = c^t_l \quad \text{and} \quad p_1(l, t - \tau) = p_1(l, t) + \tau \cdot c^t_l.$$

Let us return to our running example. We will apply the approximation p_1 to the example shown in Figure 1. We will set $\varepsilon = 0.1$, and focus on the interval $[1.1, 1.2]$ with initial values $p_1(G, 1.2) = 1$, $p_1(l, 1.2) = 0.244$, $p_1(l_R, 1.2) = 0.107$, $p_1(l_S, 1.2) = 0.075$, $p_1(\bot, 1.2) = 0$. These are close to the true values at time 1.2. Note that the point t_1, which is the time at which the reachability player switches the action played at l_R, is contained in the interval $[1.1, 1.2]$. Applying Equation (4) with these values allows us to show that the maximising action at l_R is a, and the minimising action at l_S is also a. As a result, we obtain the approximation $p_1(l_R, t - \tau) = 0.0286\tau + 0.107$ and $p_1(l_S, t - \tau) = 0.032\tau + 0.075$.

We now prove error bounds for p_1. Recall that $\mathcal{E}(1, \tau)$ denotes the difference between f and p_1 after τ time units. We can now formally define this error, and prove the following bounds.

▶ **Lemma 4.** *If $\varepsilon \leq 1$, then $\mathcal{E}(1, \varepsilon) := \| f^t_{p_1(t)}(t - \varepsilon) - p_1(t - \varepsilon) \| \leq \varepsilon^2$.*

The approximation p_1 can also be used to construct strategies for the two players with similar error bounds. We will describe the construction for the reachability player. The construction for the safety player can be derived analogously.

The strategy for the reachability player is to play the action chosen by p_1 during the entire interval $[t - \varepsilon, t]$. We will define a system of differential equations $g_1(l, \tau)$ that describe the outcome when the reachability fixes this strategy, and when the safety player plays an optimal counter strategy. For each location l, we define $g_1(l, t) = f^t_{p_1(t)}(l, t)$, and we define $g_1(l, \tau)$, for each $\tau \in [t - \varepsilon, t]$, as:

$$-\dot{g}_1(l, \tau) = \sum_{l' \in L} \mathbf{Q}(l, a^t_l, l') \cdot g_1(l', \tau) \qquad \text{if } l \in L_r, \tag{5}$$

$$-\dot{g}_1(l, \tau) = \min_{a \in \Sigma(l)} \sum_{l' \in L} \mathbf{Q}(l, a, l') \cdot g_1(l', \tau) \qquad \text{if } l \in L_s. \tag{6}$$

We can prove the following bounds for $\mathcal{E}_s(1, \varepsilon)$, which is the difference between g_1 and $f^t_{p_1(t)}$ on an interval of length ε.

▶ **Lemma 5.** *We have $\mathcal{E}_s(1, \varepsilon) := \| g_1(t - \varepsilon) - f^t_{p_1(t)}(t - \varepsilon) \| \leq 2 \cdot \varepsilon^2$.*

Lemma 4 gives the ε-step error for p_1, and we can apply Lemma 3 to show that the global error is bounded by $\varepsilon^2 \cdot \frac{T}{\varepsilon} = \varepsilon T$. If π is the required precision, then we can choose $\varepsilon = \frac{\pi}{T}$ to produce an algorithm that terminates after $\frac{T}{\varepsilon} \approx \frac{T^2}{\pi}$ many steps. Hence, we obtain the following known result.

▶ **Theorem 6.** *For a normed Markov game $\mathcal{M}$ of size $|\mathcal{M}|$, we can compute a π-optimal strategy and determine the quality of $\mathcal{M}$ up to precision π in time $O(|\mathcal{M}| \cdot T \cdot \frac{T}{\pi})$.*

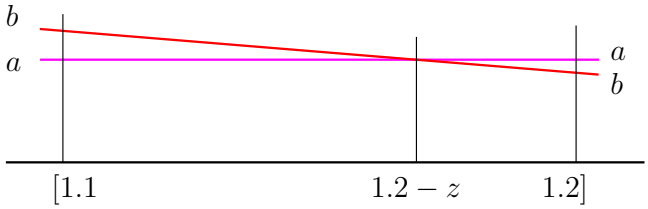


Figure 2 This figure shows how $-\dot{p}_2$ is computed on the interval $[1.1, 1.2]$ for the location l_R. The function is given by the upper envelope of the two functions: it agrees with the quality of a on the interval $[1.2 - z, 1.2]$ and with the quality of b on the interval $[1.1, 1.2 - z]$.

3.3 Double ε-Nets

In this section we show that only a small amount of additional computation effort needs to be expended in order to dramatically improve over the precision obtained by single ε-nets. This will allow us to use much larger values of ε while still retaining our desired precision.

In single ε-nets, we computed the gradient of f at the start of each interval and assumed that the gradient remained constant for the duration of that interval. This gave us the approximation p_1. The key idea behind double ε-nets is that we can use the approximation p_1 to approximate the gradient of f throughout the interval.

We define the approximation p_2 as follows: we have $p_2(l, T) = 1$ if $l \in G$ and 0 otherwise, and if $p_2(l, \tau)$ is defined for every $l \in L$ and every $\tau \in [t, T]$, then we define $p_2(l, \tau)$ for every $\tau \in [t - \varepsilon, t]$ as:

$$-\dot{p}_2(l, \tau) = \operatorname*{opt}_{a \in \Sigma(l)} \sum_{l' \in L} \mathbf{R}(l, a, l') \cdot (p_1(l', \tau) - p_1(l, \tau)) \quad \forall l \in L. \tag{7}$$

By comparing Equations (7) and (2), we can see that double ε-nets uses p_1 as an approximation for f during the interval $[t - \varepsilon, t]$. Furthermore, in contrast to p_1, note that the approximation p_2 can change it's choice of optimal action during the interval. The ability to change the choice of action during an interval is the key property that allows us to prove stronger error bounds than previous work.

▶ **Lemma 7.** If $\varepsilon \leq 1$ then $\mathcal{E}(2, \varepsilon) := \|p_2(\tau) - f^t_{p_2(t)}(\tau)\| \leq \frac{2}{3}\varepsilon^3$.

Let us apply the approximation p_2 to the example shown in Figure 1. We will again use the interval $[1.1, 1.2]$, and we will use initial values that were used when we applied single ε-nets to the example in Section 3.2. We will focus on the location l_R. From the previous section, we know that $p_1(l_R, t - \tau) = 0.0286\tau + 0.107$, and for the actions a and b we have:

- $\sum_{l' \in L} \mathbf{R}(l_R, a, l') p_1(l', t - \tau) = \frac{1}{20} + \frac{4}{5} p_1(l_R, t - \tau)$,
- $\sum_{l' \in L} \mathbf{R}(l_R, b, l') p_1(l', t - \tau) = \frac{1}{5} p_1(l, t - \tau) + \frac{4}{5} p_1(l_R, t - \tau)$.

These functions are shown in Figure 2. To obtain the approximation p_2, we must take the maximum of these two functions. Since p_1 is a linear function, we know that these two functions have exactly one crossing point, and it can be determined that this point occurs when $p_1(l, t - \tau) = 0.25$, which happens at $\tau = z := \frac{5}{63}$. Since $z \leq 0.1 = \varepsilon$, we know that the lines intersect within the interval $[1.1, 1.2]$. Consequently, we get the following piecewise quadratic function for p_2:

- When $0 \leq \tau \leq z$, we use the action a and obtain $-\dot{p}_2(l_R, t - \tau) = -0.00572\tau + 0.0286$, which implies that $p_2(l_R, t - \tau) = -0.00286\tau^2 + 0.0286\tau + 0.107$.
- When $z < \tau \leq 0.1$ we use action b and obtain $-\dot{p}_2(l_R, t - \tau) = 0.0094\tau + 0.0274$, which implies that $p_2(l_R, t - \tau) = 0.0047\tau^2 + 0.0274\tau + 0.107047619$.

As with single ε-nets, we can provide a strategy that obtains similar error bounds. Once again, we will consider only the reachability player, because the proof can easily be generalised for the safety player. In much the same way as we did for g_1, we will define a system of differential equations $g_2(l,\tau)$ that describe the outcome when the reachability player plays according to p_2, and the safety player plays an optimal counter strategy. For each location l, we define $g_2(l,t) = f^t_{p_2(t)}(l,t)$. If a^τ_l denotes the action that maximises Equation (7) at the time point $\tau \in [t-\varepsilon, t]$, then we define $g_2(l,\tau)$, as:

$$-\dot{g_2}(l,\tau) = \sum_{l' \in L} \mathbf{Q}(l, a^\tau_l, l') \cdot g_2(l',\tau) \qquad \text{if } l \in L_r, \tag{8}$$

$$-\dot{g_2}(l,\tau) = \min_{a \in \Sigma(l)} \sum_{l' \in L} \mathbf{Q}(l, a, l') \cdot g_2(l',\tau) \qquad \text{if } l \in L_s. \tag{9}$$

The following lemma proves that difference between g_2 and $f^t_{p_2(t)}$ has similar bounds to those shown in Lemma 7

▶ **Lemma 8.** *If $\varepsilon \leq 1$ then we have $\mathcal{E}_s(2,\varepsilon) := \|g_2(t-\varepsilon) - f^t_{p_2(t)}(t-\varepsilon)\| \leq 2 \cdot \varepsilon^3$.*

Computing the approximation p_2 for an interval $[t-\varepsilon, t]$ is not expensive. The fact that p_1 is linear implies that each action can be used for at most one subinterval of $[t-\varepsilon, t]$. Therefore, there are less than $|\Sigma|$ points at which the strategy changes, which implies that p_2 is a piecewise quadratic function with at most $|\Sigma|$ pieces. It is possible to design an algorithm that uses sorting to compute these switching points, achieving the following complexity.

▶ **Lemma 9.** Computing p_2 for an interval $[t-\varepsilon, t]$ takes $O(|\mathcal{M}| + |L| \cdot |\Sigma| \cdot \log|\Sigma|)$ time.

Since the ε-step error for double ε-nets is bounded by ε^3, we can apply Lemma 3 to conclude that the global error is bounded by $\varepsilon^3 \cdot \frac{T}{\varepsilon} = \varepsilon^2 T$. Therefore, if we want to compute f with a precision of π, we should choose $\varepsilon \approx \sqrt{\frac{\pi}{T}}$, which gives $\frac{T}{\varepsilon} \approx \frac{T^{1.5}}{\sqrt{\pi}}$ distinct intervals.

▶ **Theorem 10.** For a normed Markov game $\mathcal{M}$ we can approximate the time-bounded reachability, construct π optimal memoryless strategies for both players, and determine the quality of these strategies with precision π in time $O(|\mathcal{M}| \cdot T \cdot \sqrt{\frac{T}{\pi}} + |L| \cdot T \cdot \sqrt{\frac{T}{\pi}} \cdot |\Sigma| \log|\Sigma|)$.

3.4 Triple ε-Nets and Beyond

The techniques used to construct the approximation p_2 from the approximation p_1 can be generalised. This is because the only property of p_1 that is used in the proof of Lemma 7 is the fact that it is a piecewise polynomial function that approximates f. Therefore, we can inductively define a sequence of approximations p_k as follows:

$$-\dot{p_k}(l,\tau) = \operatorname{opt}_{a \in \Sigma(l)} \sum_{l' \in L} \mathbf{R}(l,a,l') \cdot (p_{k-1}(l',\tau) - p_{k-1}(l,\tau)) \tag{10}$$

We can repeat the arguments from the previous sections to obtain the following error bounds:

▶ **Lemma 11.** For every $k > 2$, if we have $\mathcal{E}(k,\varepsilon) \leq c \cdot \varepsilon^{k+1}$, then we have $\mathcal{E}(k+1,\varepsilon) \leq \frac{2}{k+2} \cdot c \cdot \varepsilon^{k+2}$. Moreover, if we additionally have that $\mathcal{E}_s(k,\varepsilon) \leq d \cdot \varepsilon^{k+1}$, then we also have that $\mathcal{E}_s(k+1,\varepsilon) \leq \frac{8c+3d}{k+2} \cdot \varepsilon^{k+2}$.

Computing the accuracies explicitly for the first four levels of ε-nets gives:

k	1	2	3	4	...
$\mathcal{E}(k,\varepsilon)$	ε^2	$\frac{2}{3}\varepsilon^3$	$\frac{1}{3}\varepsilon^4$	$\frac{2}{15}\varepsilon^5$	...
$\mathcal{E}_s(k,\varepsilon)$	$2\varepsilon^2$	$2\varepsilon^3$	$\frac{17}{6}\varepsilon^4$	$\frac{67}{30}\varepsilon^5$	...

We can also compute, for a given precision π, the value of ε that should be used in order to achieve an accuracy of π with ε-nets of level k.

▶ **Lemma 12.** To obtain a precision π with an ε-net of level k, we choose $\varepsilon \approx \sqrt[k]{\frac{\pi}{T}}$, resulting in $\frac{T}{\varepsilon} \approx T\sqrt[k]{\frac{T}{\pi}}$ steps.

Unfortunately, the cost of computing ε-nets of level k becomes increasingly prohibitive as k increases. To see why, we first give a property of the functions p_k. Recall that p_2 is a piecewise quadratic function. It is not too difficult to see how this generalises to the approximations p_k.

▶ **Lemma 13.** *The approximation p_k is piecewise polynomial with degree less than or equal to k.*

Although these functions are well-behaved in the sense that they are always piecewise polynomial, the number of pieces can grow exponentially in the worst case. The following lemma describes this bound.

▶ **Lemma 14.** If p_{k-1} has c pieces in the interval $[t-\varepsilon, t]$, then p_k has at most $\frac{1}{2}\cdot c\cdot k\cdot|L|\cdot|\Sigma|^2$ pieces in the interval $[t-\varepsilon, t]$.

The upper bound given above is quite coarse, and we would be surprised if it were found to be tight. Moreover, we do not believe that the number of pieces will grow anywhere close to this bound in practice. This is because it is rare, in our experience, for optimal strategies to change their decision many times within a small time interval.

However, there is a more significant issue that makes ε-nets become impractical as k increases. In order to compute the approximation p_k, we must be able to compute the roots of polynomials with degree $k-1$. Since we can only efficiently compute the roots of quadratic functions, and efficiently approximate the roots of cubic functions, only the approximations p_3 and p_4 are realistically useful.

Once again it is possible to provide a smart algorithm that uses sorting in order to find the switching points in the functions p_3 and p_4, which gives the following bounds on the cost of computing them.

▶ **Theorem 15.** For a normed Markov $\mathcal{M}$ we can construct π optimal memoryless strategies for both players and determine the quality of these strategies with precision π in time $O(|L|^2\cdot\sqrt[3]{\frac{T}{\pi}}\cdot T\cdot|\Sigma|^4\log|\Sigma|)$ when using triple ε-nets, and in time $O(|L|^3\cdot\sqrt[4]{\frac{T}{\pi}}\cdot T\cdot|\Sigma|^6\log|\Sigma|)$ when using quadruple ε-nets.

It is not clear if triple and quadruple ε-nets will only be of theoretical interest, or if they will be useful in practice. It should be noted that the worst case complexity bounds given by Theorem 15 arise from the upper bound on the number of switching points given in Lemma 14. Thus, if the number of switching points that occur in practical examples is small, these techniques may become more attractive. Our experiments in the following section give some evidence that this may be true.

4 Experimental Results and Conclusion

In order to test the practicability of our algorithms, we have implemented both double and triple-ε nets. We evaluated these algorithms on two sets of examples. Firstly, we tested our algorithms on the Erlang-example (see Figure 3) presented in [4] and [15]. We chose

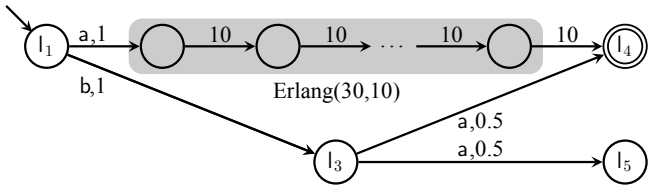


Figure 3 A CTMDP offering the choice between a long chain of fast transition and a slower path that looses some probability mass in l_5.

Table 2 Experimental evalutation of our algorithms.

	Erlang model			Game model	
precision \ method	MRMC [4]	Double-nets	Triple-nets	Double-nets	Triple-nets
10^{-4}	0.05 s	0.04 s	0.01 s	0.29 s	0.06 s
10^{-5}	0.20 s	0.10 s	0.02 s	0.93 s	0.13 s
10^{-6}	1.32 s	0.32 s	0.03 s	2.94 s	0.28 s
10^{-7}	8 s	0.98 s	0.06 s	9.35 s	0.60 s
10^{-8}	475 s	3.11 s	0.12 s	29.21 s	1.29 s
10^{-9}	—	9.91 s	0.27 s	94 s	2.78 s
10^{-10}	—	31.24 s	0.58 s	299 s	6.05 s

to consider the same parameters used by those papers: we consider maximal probability to reach location l_4 from l_1 within 7 time units. Since this example is a CTMDP, we were able to compare our results with the Markov Reward Model Checker (MRMC) [4] implementation, which includes an implementation of the techniques proposed by Buckholz and Schulz.

We also tested our algorithms on continuous-time Markov games, where we used the model depicted in Figure 4, consisting of two chains of locations $l_1, l_2, \ldots, l_{100}$ and $l'_1, l'_2, \ldots, l'_{100}$ that are controlled by the maximising player and the minimising player, respectively. This example is designed to produce a large number of switching points. In every location l_i of the maximising player, there is the choice between the short but slow route along the chain of maximising locations, and the slightly longer route which uses the minimising player's locations. If very little time remains, the maximising player prefers to take the slower actions, as fewer transitions are required to reach the goal using these actions. The maximiser also prefers these actions when a large amount of time remains. However, between these two extremes, there is a time interval in which it is advantageous for the maximising player to take the action with rate 3. A similar situation occurs for the minimising player, and this leads to a large number of points where the players change their strategy.

The results of our experiments are shown in Table 2. The MRMC implementation was unable to provide results for precisions beyond $1.86 \cdot 10^{-9}$. For the Erlang examples we found that, as the desired precision increases, our algorithms draw further ahead of the current techniques. The most interesting outcome of these experiments is the validation of triple ε-nets for practical use. While the worst case theoretical bounds arising from Lemma 14 indicated that the cost of computing the approximation for each interval may become prohibitive, these results show that the worst case does not seem to play a role in practice. In fact, we found that the number of switching points summed over all intervals and locations never exceeded 2 in this example.

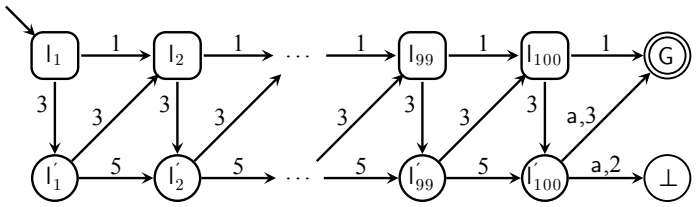


Figure 4 A CTMG with many switching points.

Our results on Markov games demonstrate that our algorithms are capable of solving non-trivially sized games in practice. Once again we find that triple ε-nets provide a substantial performance increase over double ε-nets, and that the worst case bounds given by Lemma 14 do not seem occur. Double ε-nets found 297 points where the strategy changed during an interval, and triple ε-nets found 684 such points. Hence, the $|L||\Sigma|^2$ factor given in Lemma 14 does not seem to arise here.

References

1 C. Baier, H. Hermanns, J.-P. Katoen, and B. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theoretical Computer Science*, 345(1):2–26, 2005.

2 R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

3 M. Bozzano, A. Cimatti, M. Roveri, J.-P. Katoen, V. Y. Nguyen, and T. Noll. Verification and performance evaluation of AADL models. In *ESEC/SIGSOFT FSE*, pages 285–286, 2009.

4 P. Buchholz, E. M. Hahn, H. Hermanns, and L. Zhang. Model checking algorithms for CTMDPs. In *Proc. of CAV*, pages 225–242, 2011.

5 P. Buchholz and I. Schulz. Numerical analysis of continuous time Markov decision processes over finite horizons. *Computers and Operations Research*, 38(3):651–659, 2011.

6 T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Computing maximum reachability probabilities in Markovian timed automata. Technical report, RWTH Aachen, 2010.

7 N. Coste, H. Hermanns, E. Lantreibecq, and W. Serwe. Towards performance prediction of compositional models in industrial gals designs. In *Proc. of CAV*, pages 204–218, 2009.

8 H. Garavel, R. Mateescu, F. Lang, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. In *Proc. of CAV*, pages 158–163, 2007.

9 T. A. Henzinger, M. Mateescu, and V. Wolf. Sliding window abstraction for infinite Markov chains. In *Proc. of CAV*, pages 337–352, 2009.

10 A. Martin-Löfs. Optimal control of a continuous-time Markov chain with periodic transition probabilities. *Operations Research*, 15(5):872–881, 1967.

11 B. L. Miller. Finite state continuous time Markov decision processes with a finite planning horizon. *SIAM Journal on Control*, 6(2):266–280, 1968.

12 M. R. Neuhäußer, M. Stoelinga, and J.-P. Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *Proc. of FOSSACS*, pages 364–379, 2009.

13 M. R. Neuhäußer and L. Zhang. Time-bounded reachability probabilities in continuous-time Markov decision processes. In *Proc. of QEST*, pages 209–218, 2010.

14 M. Rabe and S. Schewe. Finite optimal control for time-bounded reachability in continuous-time Markov games and CTMDPs. *Acta Informatica*, pages 291–315, 2011.

15 L. Zhang and M. R. Neuhäußer. Model checking interactive Markov chains. In *Proc. of TACAS*, pages 53–68, 2010.

Minimal Disclosure in Partially Observable Markov Decision Processes

Nathalie Bertrand[1] and Blaise Genest[2]

1 **INRIA Rennes Bretagne Atlantique, France**
2 **CNRS, UMI IPAL, joint with NUS and A*STAR/I2R, Singapore**

Abstract

For security and efficiency reasons, most systems do not give the users a full access to their information. One key specification formalism for these systems are the so called *Partially Observable Markov Decision Processes* (*POMDP* for short), which have been extensively studied in several research communities, among which AI and model-checking. In this paper we tackle the problem of the *minimal information* a user needs *at runtime* to achieve a simple goal, modeled as reaching an objective with probability one. More precisely, to achieve her goal, the user can at each step either choose to use the partial information, or pay a fixed cost and receive the full information. The natural question is then to minimize the cost the user needs to fulfill her objective. This optimization question gives rise to two different problems, whether we consider to minimize the *worst case cost*, or the *average cost*. On the one hand, concerning the worst case cost, we show that efficient techniques from the model checking community can be adapted to compute the optimal worst case cost and give optimal strategies for the users. On the other hand, we show that the optimal average price (a question typically considered in the AI community) cannot be computed in general, nor can it be approximated in polynomial time even up to a large approximation factor.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, G.3 Probability and Statistics, Markov Processes

Keywords and phrases Partially Observable Markov Decision Processes, Stochastic Games, Model-Checking, Worst-Case/Average-Case Analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.411

1 Introduction

Partially Observable Markov Decision Processes (POMDP for short) form a powerful model to describe systems where part of the information is not accessible at runtime by the user, and where the effect of the user actions is randomized. Partial observation happens virtually in every real life systems for various reasons, *e.g.* complexity, privacy or security. The usual question is given the observation at runtime and the (offline) POMDP description of the complete system, can a user achieve some goal or optimize some value?

In this paper, rather than considering that the partial information is rigidly fixed, we aim at evaluating several observation schemes. In applications where partial observation arises from complexity reasons, the system should provide *at runtime* the weakest observation which still allows to achieve a given goal, as it would also have the minimum cost of deployment. On the contrary, in the context of security, the objective is to design a secure system preventing an attacker to achieve her goal, by giving her only partial access to the state of the system. Knowing that the partial observation scheme may be vulnerable, one

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 411–422

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

may be extremely careful and analyze the additional information (obtained by punctually attacking the partial observation) an attacker needs *at runtime* to achieve her objective.

We analyze in depth the simplest such framework to which many problems can be reduced: The aim of the user is to reach with probability 1 a set Goal of states. We consider only two alternatives of information: by default, the user gets a fixed partial information; If requested, she can also obtain full information on the current state of the system. In this framework, each execution is naturally assigned a "cost" – the number of times the full information is requested – and the user always aims at minimizing this amount. Now, *giving a value* to a strategy reaching Goal can be done in two different meaningful ways: either the *worst case cost* the user can have to pay or the *average cost* she pays, while following this strategy. For both options, the objective is to compute the optimal cost (among almost-sure winning strategies) and if possible synthesize a family of strategies approximating this optimal for smaller and smaller approximation factors.

A first contribution is to show that efficient model-checking techniques can be adapted to compute the worse case cost. Furthermore, we design optimal strategies in such cases and prove that strategies with finite memory, based on the set $\mathcal{B}$ of so-called (discrete) belief states, are optimal. A belief state represents the set of states the system can be in after a given sequence of actions and observations. First, we check in polynomial time that Goal can be reached with probability 1 using req unrestrictedly. If it is not the case, the whole procedure is pointless. Assuming Goal can be reached almost surely, we define a family of generic strategies $(\sigma^n_{can})_{n\in\mathbb{N}}$ with memory state in $\mathcal{B}$ and which is almost-surely winning. To improve the worst case cost of these strategies, we compute the set of states from which Goal can be reached with probability 1 without ever requesting the full information, then the set of states requesting the full information at most once, at most twice etc. The whole process terminates in time polynomial in $|\mathcal{B}|$, and gives associated strategies with finite memory $\mathcal{B}$. Of course, $|\mathcal{B}|$ is at worse exponential in the number of states of the POMDP. The memory-size of these strategies, as well as the complexity of our algorithm are thus optimal since deciding whether Goal can be reached almost-surely without any requests (a simple subproblem) is EXPTIME-complete and requires strategies with exponential memory [6]. We illustrate the approach on a simple example for which we analyze the family of strategies, and show that it is optimal also for average cost on this particular instance.

However, this optimality result for average cost in this simple example is far from being generally true. Indeed, first of all, we prove that computing optimal average cost for a POMDP under a reachability objective is undecidable. Even worse, it is undecidable to even approximate it, whatever the approximation factor. At last, we give non approximability factors exponential in the size of the system, and prove that there is no algorithm running in polynomial time in the number of belief states to approximate the optimal average cost within that factor, even over finite horizon, for which the undecidability result does not hold.

Related work

The POMDP model has been studied by at least two communities: first by the Artificial Intelligence and Operations Research community where mostly the problem consists in optimizing a reward function. Here, the results are twofold. First, they propose heuristics to obtain policies to get good rewards, using value iteration, grid-based algorithms [11], strategy improvement [10], etc.; see for instance [14] for a survey. Also, they analyze the complexity class in which such problem falls: It is undecidable in general to compute or approximate the optimal reward [13], and it is NP-complete to do it in finite horizon [12]. Compared to these works, we consider a particular cost function which cannot be expressed

as a reward, and our lower bound results needs only a polynomial number of belief states.

More recently, the Model Checking community considered POMDP, where the question was mainly qualitative (probability 1 or positive) over a navigational goal: reachability or safety (avoiding to reach a state), once or repeatedly. The problem of reachability with probability 1 is the dual problem of safety with positive probability. These problems are EXPTIME-complete in general [6] and PSPACE-complete when the user has no observation at all [16, 4]. While visiting infinitely often a state almost-surely has the same complexity as almost sure reachability, reaching infinitely often a state with positive probability is undecidable for POMDP [1]. Surprisingly, with a slight constraint on these infinitely many visits, namely that the limit average number of times the goal is visited shall be positive, this last problem is decidable [16]. More complex systems than POMDP have been considered, where two partially informed players have opposite objectives: the results on POMDP mentionned above basically carry over [2]. More general winning conditions have also been studied [5]. Only very recently (up to our knowledge), both a particular numerical function (energy) and navigational goal were considered [9].

The closest work to ours is by Chatterjee et al [8, 7], where the problem is the existence of a controller with limited budget (and of an optimal controller) achieving an omega-regular objective in an imperfect information game. In this work and similarly to ours, partial observation of the controller is not fixed and varies according to the choices of the controller to allocate its budget. The main difference from our setting is that the arena as well as the strategy of the controller are *pure*, that is, they do not incorporate probabilities.

Controllers that dynamically request access to more precise information (e.g. by activating sensors) have also been considered in control and diagnosis for discrete event systems [15, 3].

2 Notations

Given S a finite set, let $\mathsf{Dist}(S)$ denote the set of *distributions* over S, that is functions $d : S \to [0,1]$ such that $\sum_{s\in S} d(s) = 1$. The *support* of distribution d is defined as $\mathsf{Supp}(d) = \{s \in S \mid d(s) > 0\}$. If d is the *Dirac distribution* associated with $s \in S$ (that is $d(s) = 1$ and $d(t) = 0$ for every $t \neq s$) we will abuse notation and simply write $d = s$.

▶ **Definition 1** ((PO)MDP)**.** A *Markov decision process* (MDP) is tuple $(Q, \mathsf{Act}, \Delta)$ where Q is a finite set of states, *Act* is a finite set of actions, and $\Delta : Q \times \mathsf{Act} \to \mathsf{Dist}(Q)$ is the transition function.

A *partially observable MDP* (POMDP) is a tuple $\mathcal{M} = (Q, \mathsf{Act}, \Delta, \mathsf{Part})$, where $(Q, \mathsf{Act}, \Delta)$ is an MDP, and Part is a partition of Q. Elements of Part are called *observations*.

Given a POMDP $(Q, \mathsf{Act}, \Delta, \mathsf{Part})$ the underlying MDP $(Q, \mathsf{Act}, \Delta)$ is alternatively called the FOMDP, for *Fully Observable* MDP. Intuitively, in a POMDP, from state s, if action a is chosen, the next state is t with probability $\Delta(s, a)(t)$ and the controller receives observation $O \in \mathsf{Part}$ such that $p \in O$. Of course, any MDP can be seen as a POMDP by setting Part as the set of all singletons sets. In the sequel, we always assume a fixed starting state s_0 and also distinguish a goal (set of) state(s) Goal. We assume for convenience that being in Goal is observable, that is for all $P \in \mathsf{Part}$, $P \subseteq 2^{\mathsf{Goal}} \cap P$ or $P \subseteq 2^{Q\setminus\mathsf{Goal}}$.

An example of POMDP is given in Figure 1, where $\mathsf{Part} = \big\{\{1,2,3\}, \{\mathsf{Goal}\}, \{4\}\big\}$ and the non-trivial set of the partition is represented by a grey area.

In this paper, given a POMDP $(Q, \mathsf{Act}, \Delta, \mathsf{Part})$, we assume that the controller can perform an extra action $\mathsf{req} \notin \mathsf{Act}$ whose effect is to disclose the precise state of the FOMDP.

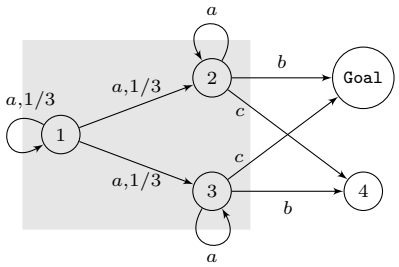


Figure 1 A simple POMDP

More precisely, the observation the controller receives after a request action req is $\{s\}$ if the current state is $s \in Q$. The set of possible observations, denoted $\mathcal{O}$, thus consists of the partition Part, together with $\mathsf{Single} = \{\{s\} \mid s \in Q\}$. The probabilistic transition function is extended to $\mathsf{Act}' = \mathsf{Act} \cup \{\mathsf{req}\}$ by defining $\Delta(s, \mathsf{req})$ as the Dirac distribution associated with observation $\{s\}$. Our high-level aim is to design a strategy of chosing actions to play based only on the sequence of actions played and observations received, such that the goal can be reached. Notice that from $s_0 = 1$, one needs to play a as no other choice is possible. A possible outcome is to reach state 1 again, hence for any choice of actions, there will always be a possibility to not reach Goal. Hence our aim is instead to ensure that Goal is reached, unless being terribly unlucky. Formally, we want the set of paths reaching Goal to have probability 1. We define the probability space of G by extending the POMDP to Act'.

2.1 Belief States

Let $(a_1, O_1) \cdots (a_i, O_i) \in (\mathsf{Act} \times \mathsf{Part} \cup \{\mathsf{req}\} \times \mathsf{Single})^*$ be a sequence of actions played and observations received. We define $B(s_0, (a_1, O_1) \cdots (a_i, O_i))$ as the set of states $s \in Q$ such that for all $0 \leq j \leq i$ there exists t_j with $t_0 = s_0$, $t_i = s$, $t_j \in O_j$ and $\Delta(t_{j-1}, a_j)(t_j) > 0$. Intuitively if actions $a_1 \cdots a_i$ have been played and observations $O_1 \cdots O_i$ were received, the set of possible states are exactly those in $B(s_0, (a_1, O_1) \cdots (a_i, O_i))$. The set $B(s_0, (a_1, O_1) \cdots (a_i, O_i))$ can be computed inductively:

$$B(s_0, (a_1, O_1) \cdots (a_i, O_i)) = O_i \cap \{t \mid \exists s \in B(s_0, (a_1, O_1) \cdots (a_{i-1}, O_{i-1})),\ \Delta(s, a_i)(t) > 0\}.$$

The set Path of *possible finite paths* consists in all $\rho = (a_1, O_1) \cdots (a_n, O_n) \in (\mathsf{Act} \times \mathsf{Part} \cup \{\mathsf{req}\} \times \mathsf{Single})^*$ such that $B(s_0, (a_1, O_1) \cdots (a_n, O_n)) \neq \emptyset$. Moreover, the set of *reachable belief states* is defined by $\mathcal{B} = \{B(s_0, \rho) \mid \rho \in \mathsf{Path}\}$. For the example POMDP of Figure 1 the graph of reachable belief states is depicted in Figure 2. The number of belief states is at worst exponential in the number of states, but often, it is not that big.

Given a finite path ρ, $B(s_0, \rho)$ only gives the set of states the POMDP can be after executing ρ, but the precise probability to be in each state is unknown. The distributions $D(s_0, \rho)$ and $D(s_0, \rho, a)$ over states after a prefix of path (that is, ρ or ρa, with $\rho \in \mathsf{Path}$ and $a \in Act'$) can be defined by induction. Assuming $\rho = \rho'(a, O)$, and $D(s_0, \rho')$ is known, $D(s_0, \rho', a)$ is defined by $D(s_0, \rho', a)(s) = \sum_t D(s_0, \rho')(t) \times \Delta(t, a)(s)$. Then, taking the observation into account yields: $D(s_0, \rho)(s) = 0$ if $s \notin O$, else $D(s_0, \rho)(s) = D(s_0, \rho', a)(s) / \sum_{s \in O} D(s_0, \rho', a)(s)$.

Of course, the discrete belief after ρ can be recovered from the probabilistic belief: $B(s_0, \rho) = \mathsf{Supp}(D(s_0, \rho))$.

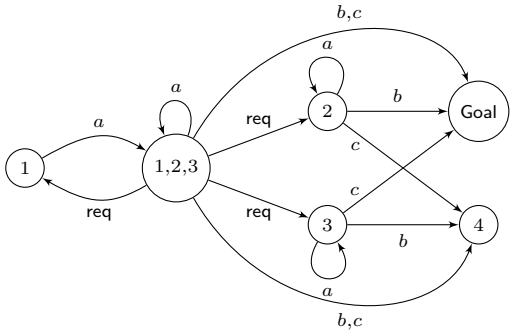


Figure 2 Graph of reachable belief states for the POMDP in Figure 1

2.2 Strategy

A *strategy* $\sigma : \mathsf{Path} \to \mathsf{Dist}(\mathsf{Act}')$ for the controller is a function which associates with any possible path a distribution over the extended set of actions Act'. Given a strategy σ, a possible path $(a_1, O_1) \cdots (a_n, O_n) \in \mathsf{Path}$ is called a *σ-path* if $a_{i+1} \in \mathsf{Supp}\big(\sigma((a_1, O_1) \cdots (a_i, O_i))\big)$ for all $i < n$. When a fixed strategy σ is played, it is easy to define the probability that a possible path occurs, as follows. First of all, if $\rho = \rho'(a, O)$ is a σ-path, $\mathbb{P}_\sigma(\rho) = \mathbb{P}_\sigma(\rho') \cdot \sigma(\rho')(a) \cdot \sum_{s \in O} D(\rho', a)(s)$, and $\mathbb{P}(\rho) = 1$ if ρ is the empty path. Now, we can define the probability of a run $r = s_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{a_n} s_n$ (every s_i is a state of the FOMDP) knowing that ρ occured. Let ρ be the unique (POMDP) path associated with r, obtained by replacing states with their associated observations. The probability of r assuming ρ is performed is given by $\mathbb{P}(r \mid \rho) = \prod_{i \leq n} D(s_0, \rho_i)(s_i)$, where ρ_i denotes the prefix of length i of ρ. Then we let $\mathbb{P}_\sigma(r) = \mathbb{P}_\sigma(\rho) \times \mathbb{P}(r \mid \rho)$. This probability measure on finite runs is extended in the usual way to the sigma-algebra they generate, and it is well-known that LTL properties on infinite runs are measurable for this measure [17]. The objective for the controller is to reach the Goal state. Thus, strategy σ is *almost-surely winning* if $\mathbb{P}_\sigma(\Diamond \mathsf{Goal}) = 1$. Clearly enough, if there is no almost-surely winning strategy in the FOMDP, then there will be none in the extended POMDP either.

Problem statement

To every strategy σ, two quantities can be associated: first of all, the worst-case cost, *i.e.* the maximum number of request actions σ takes along a σ-path; and second the average cost, that is the expected number of requests for σ-paths. In this paper we thus tackle the two distinct problems of finding almost-surely winning strategies which (1) minimize the worst-case cost, or (2) minimize the average cost.

3 Algorithms for the optimal worst case cost

Strategies are in general objects that do not have a finite presentation. In order to represent them effectively, it is common to restrict to finite-memory strategies, that are weaker than general strategies, but, as we will see, suffice when considering the optimal worst case cost problem. A *finite-memory strategy* on finite memory set M is given as $\sigma : M \to \mathsf{Dist}(\mathsf{Act}')$ together with an update function: $\mathsf{up} : M \times (\mathsf{Act} \times \mathsf{Part} \cup \{\mathsf{req}\} \times \mathsf{Single}) \to M$.

3.1 Reaching Goal with probability 1

We first propose a family $(\sigma^n_{can})_{n\in\mathbb{N}}$ of strategies with finite memory $\mathcal{B} \subseteq 2^Q$ the set of reachable belief states. The memory is initialized to $\{s_0\}$, where s_0 is the initial state. The update function is given by $\mathsf{up}(S, a, O) = O \cap \{t \mid \exists s \in S, \Delta(s, a_i)(t) > 0\}$. We extend it inductively to a path $\rho' = \rho \cdot (a, O)$ with $\mathsf{up}(S, \rho \cdot (a, O)) = \mathsf{up}(\mathsf{up}(S, \rho), a, O) = T$, and we say that the path ρ' reaches the memory state T. It is easy to see that the memory state M reached after some possible path ρ is exactly $B(s_0, \rho)$.

Let $n \in \mathbb{N}$. We now define how the strategy σ^n_{can} plays. First, we denote by $\mathsf{Lose}_F \subseteq Q$ the set of states s of the FOMDP associated with G such that there is no strategy in the FOMDP reaching Goal with probability 1 from s. We then denote by $\mathsf{Lose} \subseteq 2^Q$ the set of belief states S such that $S \cap \mathsf{Lose}_F \neq \emptyset$. In the example of Figure 1, 4 is the only state of the FOMDP from which there is no strategy reaching Goal almost-surely. In the reachable belief graph of Figure 2, Lose is thus made of the single belief state $\{4\}$. Intuitively, any path reaching a memory state in Lose cannot reach Goal with probability 1. It is easy to see that if $\{s_0\} \in \mathsf{Lose}$, then there is no almost-surely winning strategy, since otherwise a strategy in the FOMDP reproducing σ (which is possible since it has at least as much information) would also reach Goal with probability 1.

Letting $\mathsf{Win} = \mathcal{B} \setminus \mathsf{Lose}$ the complementary set, we prove in the next theorem that under strategy σ^n_{can}, Goal is reached almost-surely from any belief state of Win. We partition Win in 3 sets: $W_0 = \{S \mid S \subseteq \mathsf{Goal}\}$ wins directly, W_N the *needing* belief states, and W_U the *non-needing* belief states. A state S is in W_N whenever for all $a \in \mathsf{Act}$, there exists $O \in \mathsf{Part}$, such that $B(S, a, O) \in \mathsf{Lose}$. Intuitively, from a memory state in W_N any almost-surely winning strategy needs to perform a request action req, as every other action leaves a chance to reach Lose. In the example of Figure 2, W_0 only contains the belief state $\{\mathsf{Goal}\}$, $W_N = \emptyset$ (in particular, actions a can be done safely from belief states $\{1\}$ and $\{1, 2, 3\}$), and W_U consists of the other belief states $\{1\}, \{1, 2, 3\}, \{2\}, \{3\}$. Strategy σ^n_{can} is then defined by:

- If $M \in W_0$, then $\sigma^n_{can}(M) = \emptyset$,
- if $M \in W_N$, $\sigma^n_{can}(M) = \mathsf{req}$,
- if $M \in W_U$, $\sigma^n_{can}(M)$ plays req with probability $1/n$, and plays uniformly all actions $a \in Act$ such that for every observation O, $\mathsf{up}(M, a, O) \notin \mathsf{Lose}$, and
- if $M \in \mathsf{Lose}$, then $\sigma^n_{can}(M)$ is the uniform distributions over all actions.

Furthermore, if $M \in \mathsf{Single}$ (that is, the actual state is known for sure), then we disallow σ^n_{can} to perform a req as it would be useless. Notice that the only infinite σ^n_{can}-paths are exactly those which never meet W_0.

▶ **Theorem 2.** *If* $\{s_0\} \in \mathsf{Win}$, *then* $\mathbb{P}_{\sigma^n_{can}}(\Diamond \mathsf{Goal}) = 1$.

Notice that we can compute in polynomial time the set of losing state in the FOMDP, and hence decide in polynomial time whether there exists an almost-surely strategy in G.

3.2 Optimizing the worst case cost

Now that we know the set of belief states from which there is a strategy reaching Goal with probability 1, we can tune the canonical strategies σ^n_{can}. To do so, we compute inductively the set S_k of belief states from which one needs at most k actions req to win. The set S_0 is pretty easy to obtain, as the associated strategy cannot use any req.

Let $G_0 = (\mathcal{B}, \mathsf{Act}, \delta_0)$ be the *belief MDP* associated with the POMDP G, where each state is a belief state, and $\delta_0(B, a)$ is the uniform distribution over all belief states B' such that there exists a part $P \in \mathsf{Part}$ with $B' = P \cap \{t \mid \exists s \in B,\ \Delta(s, a)(t) > 0\}$. Notice that

requests are not allowed in G_0. The MDP G_0 obtained from the POMDP in Figure 1 is very similar to the graph of Figure 2, except that the 3 edges labeled with req have been deleted. Let us denote by S_0 the set of belief states from which there exists a strategy σ_0 in G_0 reaching $W_0 = 2^{\mathsf{Goal}}$ almost-surely. The set S_0 can be computed in time linear in the number $|\mathcal{B}|$ of states of G_0, thus at worst in time exponential in the size of G. Taking our example of Figure 1, $S_0 = \big\{\{2\},\{3\},\{\mathsf{Goal}\}\big\}$. Indeed, $\{4\}$ is a losing state, and $\{1\}$ and $\{1,2,3\}$ as well since no req-action is allowed and playing b, c from $\{1,2,3\}$ has a positive probability to lead to $\{4\}$. Clearly, σ_0 can be chosen positional, and on the example it is given by $\sigma_0(\{2\}) = b$ and $\sigma_0(\{3\}) = c$.

Now, the canonical strategy σ^n_{can} is improved by letting $\sigma^n_{can}(B) = \sigma_0(B)$ if $B \in S_0 \setminus W_0$, and leaving it unchanged otherwise. Under this new definition, σ^n_{can} is still almost-surely reaching Goal, and from any $B \in S_0$, σ^n_{can} never proposes a req-action anymore:

▶ **Proposition 3.** Assuming $s_0 \in \mathsf{Win}$, then: (i) σ^n_{can} is almost-surely winning, and (ii) for every σ^n_{can}-path $\rho = \rho_1\rho_2$ with $B(\rho_1) \in S_0$, ρ_2 contains no req.

Given a strategy σ, we say that B is a *σ-belief state* if there exists a σ-path ρ with $B(\rho) = B$. We prove that S_0 is optimal, in the following meaning:

▶ **Proposition 4.** Let σ be a strategy reaching Goal almost-surely from s_0, and $B \notin S_0$ a σ-belief state. Then there exists a σ-path $\rho = \rho_1\rho_2$ such that $B(\rho_1) = B$ and ρ_2 contains a req.

Proof. Let σ be an almost-surely winning strategy, and B a σ-belief state. Assume by contradiction that for every σ-path $\rho_1\rho_2$ with $B(\rho_1) = B$, ρ_2 contains no req-action, and let us prove that $B \in S_0$. We design a strategy σ' in the MDP G_0 from B as follows. For each run $r = Ba_1B_1 \cdots a_nB_n$ in G_0, let $\rho_r = Oa_1O_1 \cdots a_nO_n$ be the associated possible path in G with $B_i \subseteq O_i$ for all i. The choice of O_i is unique, and it is always a part of Part since $a_i \neq \mathsf{req}$. We let $\sigma'(r) = \sigma(\rho_1\rho_r)$. Now, it is easy to see that σ' reaches Goal almost surely from B, since ρ_1 is a finite σ-path and σ reaches Goal with probability 1. As a consequence, $B \in S_0$. ◀

We can now define the set S_1 of belief states for which at most one req-action is needed to reach Goal almost-surely. We let L_1 be the set of belief states B such that for all $s \in B$, $\{s\} \in S_0$. Playing a request from a state in L_1 obviously leads to some state in S_0, from which winning without request is possible. Clearly, $L_1 \cup S_0 \subseteq S_1$. In fact, S_1 is the set of all belief states from which there is a strategy to reach $L_1 \cup S_0$ in G_0 with probability one. S_1 can be computed as follows:

1. Initialize a set X of belief states at $\mathcal{B} \setminus \mathsf{Lose}$,
2. Compute the set $Y_X \subseteq \mathcal{B}$ of belief states which can reach $L_1 \cup S_0$ while staying in X, using a smallest fixed point:
   a. initialize Y_X at $L_1 \cup S_0$,
   b. add to Y_X all $B \in X$ such that there exists $a \in \mathsf{Act}$ with $\mathsf{Supp}(\delta_0(B,a)) \subseteq X$ and $\mathsf{Supp}(\delta_0(B,a)) \cap Y_X \neq \emptyset$.
3. If $X \neq Y_X$ then set $X := Y_X$ and goto step 2 again, else set $S_1 = Y_X$ and quit.

The so-computed set S_1 of belief states is the largest one such that from all belief states of S_1 there is a strategy which allows to reach $L_1 \cup S_0$ and which ensures to stay in S_1. Now, we improve once again the canonical strategy σ^n_{can} with,

- if $B \in L_1 \setminus S_0$, $\sigma^n_{can}(B) = \mathsf{req}$,

- if $B \in S_1 \setminus (L_1 \cup S_0)$, $\mathsf{Supp}(\sigma^n_{can}(B))$ is the uniform distribution over the set of actions $a \in \mathsf{Act}$ such that $\mathsf{Supp}(\delta_0(B,a)) \subseteq S_1$,
- otherwise, it is unchanged.

Notice that from $S_1 \setminus (L_1 \cup S_0)$, always at least one action $a \in \mathsf{Act}$ satisfies $\delta_0(B,a) \subseteq S_1$.

After this modification, σ^n_{can} still reaches Goal with probability 1 and from any $B \in S_1 \setminus S_0$, σ^n_{can} proposes at most one req-action:

▶ **Proposition 5.** Assuming $s_0 \in \mathsf{Win}$, then: (i) σ^n_{can} is almost-surely winning, and (ii) for every σ^n_{can}-path $\rho = \rho_1\rho_2$ with $B(\rho_1) \in S_1$, ρ_2 contains at most one req.

Proof. (ii) is fairly easy to establish. Let $\rho = \rho_1\rho_2$ be a σ^n_{can}-path with $B = B(\rho_1) \in S_1$. As $\rho_2 = B \xrightarrow{a_1} B_1\rho_2'$ is a σ^n_{can}-path, the first action a_1 ensures to stay in S_1: more precisely, if $B \in L_1$, then $B_1 \in S_0$, if $B \in S_0$, then $B_1 \in S_0$, and if $B \in S_1 \setminus (L_1 \cup S_0)$ then $B_1 \in S_1$. Iterating this argument, beliefs in ρ_2 always belong to S_1. In case L_1 is never reached, request are never performed. Otherwise, as soon as it reachs L_1, a request is played, and the next belief state is in S_0 from which no request are proposed anymore, according to the Proposition 3. Overall, (ii) is verified.

We now prove that from S_1, $L_1 \cup S_0$ is reached with probability 1, which proves that σ^n_{can} reaches Goal with probability 1 thanks to Proposition 3. The only paths from S_1 which does not reach $L_1 \cup S_0$ are those staying forever in $S_1 \setminus (L_1 \cup S_0)$. We now prove that from every state in S_1, there is a σ^n_{can}-path reaching $L_1 \cup S_0$, hence, a positive probability to reach $L_1 \cup S_0$. Together, these facts show that almost-surely $L_1 \cup S_0$ will be reached from S_1. Assume now by contradiction that there is a state $B \in S_1$ with $B(\rho_1) = B$ and such that for all σ^n_{can}-path $\rho = \rho_1\rho_2$, we have $B(\rho) \notin L_1 \cup S_0$. Hence these paths ρ_2 make no request. But this set of ρ_2 paths cover exactly the set of path from B which stay within S_1. Considering the last iteration of the construction of S_1. As it is the last iteration, $Y_X = X = S_1$ and $B \in X$. Now, in the construction of Y_X, B can never be added to Y_X. Hence $Y_X \neq X$, a contradiction. ◀

We prove that S_1 is optimal, in the following sense:

▶ **Proposition 6.** Let σ be a strategy reaching Goal almost-surely from s_0 and $B \notin S_1$ a σ-belief state. Then there is a σ-path $\rho = \rho_1\rho_2$ with $B(\rho_1) = B$ and ρ_2 contains at least two req-actions.

We can easily construct in this way by induction the sets $(S_k)_{k\in\mathbb{N}}$ of belief states requiering at most k requests, until stabilization: $S_{K+1} = S_K$, which happens at worse for $K = |\mathcal{B}|$. Computing each S_i takes $O(|\mathcal{B}|^2 \times |\mathsf{Act}|)$; overall, the procedure is in time $O(|\mathcal{B}|^3 \times |\mathsf{Act}|)$.

This easily improves the strategy σ^n_{can} by allowing requests only when they are needed. Then, denoting $S_\infty = 2^Q \setminus S_K$, we have the following optimality result:

▶ **Proposition 7.** For every strategy σ reaching goal almost surely and every σ-belief states $B \in S_\infty$, for all $N \in \mathbb{N}$, there exists a σ-path $\rho = \rho_1\rho_2$ with $B(\rho_1) = B$ and ρ_2 contains at least N req-actions.

In our example, notice that $S_1 = S_0$, and hence $S_\infty = \big\{\{1\},\{1,2,3\}\big\}$. Hence, the improved canonical strategy in our running example is defined by: $\sigma^n_{can}(1) = a, \sigma^n_{can}(2) = b, \sigma^n_{can}(3) = c$ and $\sigma^n_{can}(\{1,2,3\})$ assigns probability $(n-1)/n$ to action a and probability $1/n$ to req. We show now that the family of canonical strategies is, on this particular example, also optimal for average cost! The proof is educational to understand that even with a fixed strategy, computing the average cost in a POMDP is not easy as the set of possible stochastic belief states (precise distributions over states in the discrete belief states) is potentially infinite.

Let $n \in \mathbb{N}$ and consider strategy σ^n_{can}. The game starts in state 1, and σ^n_{can} first decision is to perform an a. After this action, the discrete belief state is $\{1,2,3\}$ and the probability to be in state 1 is $1/3$. In the sequel, we denote by E_k the expected number of requests following σ^n_{can} from $\{1,2,3\}$ with probability $1/3^k$ to be in state 1. Thanks to the observation above, the expected number of requests for σ^n_{can} is exactly E_1.

Assume now that the current belief state is $\{1,2,3\}$ and the probability to be in state 1 is $1/3^k$. In this state, with probability $1/n$, a request is performed, which discloses state 1 with probability $1/3^k$; with probability $(n-1)/n$, a is played, and the resulting state is $\{1,2,3\}$ with probability $1/3^{k+1}$ to be in state 1. As a consequence, $E_k = \frac{1}{n}\big(1 + \frac{1}{3^k}E_1\big) + \frac{n-1}{n}E_{k+1}$. By summation, we derive: $E_1 = 1 + \frac{1}{2n}$. Hence, the average number of req asked by σ^n_{can} is smaller than $1 + \frac{1}{2n}$, for all $n \in \mathbb{N}$. In fact, we can prove that this family of strategy is optimal in the sense that no almost surely winning strategy can achieve an average number of request of 1 or less on this particular example.

4 Hardness and undecidability results for the average cost

We turn now to a more general analysis of the strategies minimizing the average cost. Unfortunately, as we hinted before, this question is very hard to tackle. We show first that the problem of the existence of a strategy with cost smaller than a fixed threshold is in general undecidable, and that it is undecidable to approximate the optimal average cost. Moreover, we give concrete approximation factor (with respect to the size of the POMDP) up to which no optimal strategy can be computed with polynomial time algorithms. This obviously contrasts with the rather efficient algorithms we design in the previous section, which run in time polynomial in $|\mathcal{B}|$.

For a run r of G, we denote by $val(r)$ the number of req in r. Let σ be a strategy reaching $Goal$ from s_0 with probability 1. Then we denote by $val(\sigma)$ the expected value of $val(r)$, over all the σ-runs. Notice that $val(\sigma) < \infty$ since σ is almost-surely winning.

▶ **Definition 8.** The *value* of G is $val(G) = \inf\{val(\sigma) \mid \sigma \text{ almost-surely winning}\}$, where $val(\sigma)$ denotes the expected number of requests under strategy σ.

We can now present our first negative result, namely that it is undecidable to compute the exact minimum average cost $val(G)$. This result should not be too surprising as optimizing a cost function in a POMDP is undecidable [13]. However, our result is stronger and harder to get, since our cost function is not arbitrary.

▶ **Theorem 9.** *For all $K > 0$, it is undecidable to know whether $val(G) \leq K$.*

Proof. Let $\varepsilon \in (0, 1/2)$. Take a Probabilistic Finite Automaton $\mathcal{P}$ (a PFA for short, that is a POMDP with $|\mathsf{Part}| = 1$) such that either there exists a word accepted with probability at least $1-\varepsilon$ or all words are accepted with probability less than ε. It is undecidable to know which case holds [13]. From $\mathcal{P}$, we build a POMDP G, as illustrated on Figure 3 by adding four new states. This reduction ensures the following: $\mathcal{P}$ accepts a word with probability greater than $1-\varepsilon$ if and only if $val(G) < \frac{\varepsilon}{1-\varepsilon}$. ◀

Actually, in the proof, we even show that if $val(G) \geq \varepsilon/(1-\varepsilon)$, then $val(G) \geq 1-\varepsilon$. That is, $(\varepsilon/(1-\varepsilon) + 1 - \varepsilon)/2$ is the best approximation one can make for this family of POMDP. The approximation factor is thus $\delta = (1/(1-\varepsilon) - \varepsilon)(1-\varepsilon)/2\varepsilon = (1 - \varepsilon(1-\varepsilon))/2\varepsilon$, which converges to infinity as ε converges to 0. As a consequence:

▶ **Corollary 10.** *For any δ, it is undecidable to approximate $val(G)$ with factor δ.*

Notice however that for bigger δ, the non-approximation result uses bigger and bigger PFA (and thus POMDP). The following result establishes the relationship between the number of states of the POMDP and the non-approximation factor.

▶ **Theorem 11.** *Assuming that $P \neq NP$, for any polynomial time algorithm $\mathcal{A}$, there exists a POMDP G with at most $3n^2 + n$ states and at most $9n^2 + 8n$ reachable belief states, such that $\mathcal{A}$ computes a strategy with value val on G with:*
approximation factor: $|val - val(G)|/val(G) \geq 2^{n-1}/n^2 - 1$, *and*
absolute approximation error: $|val - val(G)| \geq (2^{n-1}/n - n - 2)$.

Proof. The proof is by reduction from 3-SAT. Let φ be a Boolean formula in conjunctive normal form with m clauses $C_1 \cdots C_m$ over k variables $x_1, \cdots x_k$. We let $n = k \times m$. We also let $\varepsilon = 1/2^n$. From φ, we derive a POMDP G such that φ is satisfiable if and only $val(G) \leq 3n\varepsilon$. We recall that φ is satisfiable if and only if for every clause C_i, one can choose one literal ℓ_i among the three literals of C_i such that for all $i, j \leq m$, the choices of ℓ_i and ℓ_j do not conflict.

The actions that can be played are any of the $2k$ literals $\{x_i, \overline{x_i} \mid i \leq k\}$, plus a dummy action. The POMDP starts in the initial state init, where only the dummy action can be fired, and with equal probability a variable $x_i, i \leq k$ is chosen leading to state labeled (C_1, i). Intuitively, the POMDP will remember actions played concerning this variable x_i and no other. All states $(C_1, 1), \cdots, (C_1, k)$ belong to the same part of the partition and thus the player does not know which variable is monitored.

From (C_1, i), three actions are enabled, corresponding to the three literals in clause C_1. If the action played is $x_j, \overline{x_j}$ with $j \neq i$, then the next state is (C_2, i) with probability $1 - \varepsilon$. With probability $\varepsilon/2$ it is (C_2, x_i) and with probability $\varepsilon/2$ it is $(C_2, \overline{x_i})$. Intuitively, with small probability, the POMDP remembers wrongly that literal x_i or $\overline{x_i}$ has been chosen. If the action played is x_i, then the next state is (C_2, x_i) with probability $1 - \varepsilon$, (C_2, i) with probability $\varepsilon/2$ and $(C_2, \overline{x_i})$ with probability $\varepsilon/2$. At last, if the action played is $\overline{x_i}$, then the next state is $(C_2, \overline{x_i})$ with probability $1 - \varepsilon$, (C_2, i) with probability $\varepsilon/2$ and (C_2, x_i) with probability $\varepsilon/2$. Transitions from (C_ℓ, i) with $\ell < m$ and $i \leq k$ follow the same pattern.

Now, assume that the state is $(C_\ell, \overline{x_i})$, that is the POMDP recalls (possibly wrongly) that $\overline{x_i}$ has been played before. If the action played is x_i, then there is a conflict and the next state is test, which is the gadget, similar to the one in the proof of Theorem 9 that forces the player to perform a req-action in order not to lose, and then it reaches state ok. If the action played is any other, then the next state is $(C_{\ell+1}, \overline{x_i})$ with probability $1 - \varepsilon$, and $(C_{\ell+1}, x_i)$ or $(C_{\ell+1}, i)$ with probability $\varepsilon/2$ each. From state (C_ℓ, x_i), the transitions

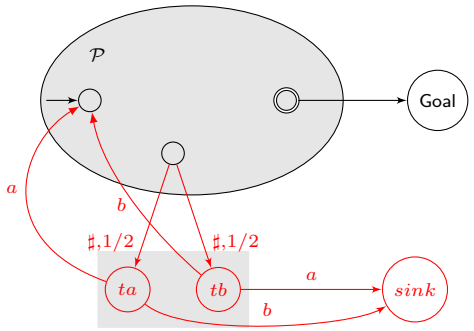


Figure 3 Reduction to a variant of the emptiness problem for PFA

are symmetric. At last, for the last clause, from (C_m, x_i), all the actions lead to ok except action $\overline{x_i}$ which leads to the test gadget.

This reduction ensures: there exists a strategy σ reaching ok with probability 1 and such that $val(\sigma) \leq n\varepsilon$ if and only if the formula φ is satisfiable. Assume that there is a non conflicting choice of literals for every clause. Consider the strategy σ which chooses to play accordingly to this choice of literal, and when in the test gadget performs a req. With probability higher than $1 - n\varepsilon$, the POMDP remembers accurately the choice of literal x_i by σ at each step (there are less than n steps). As this choice is not conflicting, under this hypothesis, no req is played and ok is reached. Now, with probability less than $n\varepsilon$, the memory can be wrong at some point, and the worse case is to reach the test, in which case a unique requests is made before reaching ok. Thus, $val(\sigma) \leq n\varepsilon$. For the reverse implication, observe that under any strategy σ reaching ok with probability 1, there is at least one conflicting variable (successive choices of literal x_i and $\overline{x_i}$). As in the reduction of Theorem 9 we assume without loss of generality that σ does not propose any req but in test. With probability at least $1/n$, the POMDP remembers the conflicting variable, and with probability at least $(1 - n\varepsilon)$ the POMDP remembers accurately the first literal of x_i played, and then when $\overline{x_i}$ is played, the POMDP goes to test where a req is played. Overall, $val(\sigma) \geq (1/n - \varepsilon)$.

As SAT is NP-complete and assuming that $P \neq NP$, no polynomial time algorithm can decide whether $val(G) \geq (1/n - \varepsilon)$ or $val(G) \leq n\varepsilon$. To keep the error factor as low as possible, the safest is thus to play the average value, which proves the first item.

One can however notice that even if the approximation factor is large, the absolute gap between an approximation and the real value is rather small (less than 1), which would not be a huge concern in practice. We explain now how to keep the same factor while widening the absolute gap using a small trick. The previous reduction is enriched as follows: from state ok only the dummy action can be played, leading with probability ε to Goal, and with probability $1 - \varepsilon$ back to init. The probability to reach Goal after seeing init exactly i times is $\varepsilon(1-\varepsilon)^i$, that is the expected number of times init is seen is $\sum_i i \cdot \varepsilon \cdot (1-\varepsilon)^i \in [2^n - 2, 2^n]$ for n large enough. That is, if there is a non conflicting choice of literals, then $val(G) \leq n/2^n \cdot 2^n = n$. On the other hand, if all choice of 1 literal per clause is conflicting, then $val(G) \geq (1/n - \varepsilon) \cdot (2^n - 2) \geq 2^n/n - 2$. This concludes the proof. ◀

Notice that the first item of our result holds for POMDP without loops, that is in particular for finite horizon POMDP (horizon $\leq n$). Compared with other results on non approximability of optimal cost in (finite horizon) POMDP [12], our reduction does not rely on the (at worse exponentially many) discrete belief states to encode the problem, but uses the actual probability to be in a state. Indeed, the family of graph considered has a polynomial number of reachable belief states (for which the algorithm of the previous section are efficient). Actually, if we relax the constraint of a polynomial number of beliefs, the proof can easily be simplified to obtain an infinite non approximability factor, since $val(G) = 0$ if and only if the 3-SAT formula is satisfiable. It also proves that reasoning on beliefs is in some sense mandatory.

5 Conclusion

In this paper we investigated the problem of minimizing requests for full information in a POMDP in order to achieve a reachability objective with probability 1. On the one hand, the optimal worst-case cost is in $\mathbb{N} \cup \{\infty\}$ and can be computed in polynomial time in the number of discrete beliefs (that is, exponential time at worse), together with a finite memory

strategy that guarantees this optimal cost. On the other hand, the optimal average cost is in $\mathbb{R}_{\geq 0}$ and can neither be computed nor approximated, and we provide large error factors for which no polynomial-time algorithm can approximate the average optimal cost up to that factor. In practice, despite the non-approximability result, quite accurate values can be obtained for some POMDPs using heuristics [14]. Our model can be enriched to allow several intermediate information levels, encoded by successive refinements of the partition, and for which the techniques developped here will certainly be useful.

Acknowledgements We would like to thank the CNRS PEPS UMI INSIS AABS for the financial support without which this work would not have taken place, as well as the referees for pointing out very interesting related bibliography.

References

1 C. Baier, N. Bertrand, and M. Grösser. On decision problems for probabilistic Büchi automata. In *FOSSACS '08*, volume 4962 of *LNCS*, pages 287–301. Springer, 2008.
2 N. Bertrand, B. Genest, and H. Gimbert. Qualitative determinacy and decidability of stochastic games with signals. In *LICS '09*, pages 319–328. IEEE, 2009.
3 F. Cassez and S. Tripakis. Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae*, 88(4):497–540, 2008.
4 R. Chadha, A. P. Sistla, and M. Viswanathan. Power of randomization in automata on infinite strings. In *CONCUR '09*, volume 5710 of *LNCS*, pages 229–243. Springer, 2009.
5 K. Chatterjee and L. Doyen. The Complexity of Partial-Observation Parity Games. In *LPAR '10 (Yogyakarta)*, LNCS 6397, pages 1–14, 2010.
6 K. Chatterjee, L. Doyen, and T. A. Henzinger. Qualitative analysis of partially-observable Markov decision processes. In *MFCS '10*, volume 6281 of *LNCS*, pages 258–269. Springer, 2010.
7 K. Chatterjee and R. Majumdar. Minimum attention controller synthesis for omega-regular objectives. In *FORMATS '11*, volume 6919 of *LNCS*, pages 145–159. Springer, 2011.
8 K. Chatterjee, R. Majumdar, and T. A. Henzinger. Controller synthesis with budget constraints. In *HSCC '08*, volume 4981 of *LNCS*, pages 72–86. Springer, 2008.
9 A. Degorre, L. Doyen, R. Gentilini, J.-F. Raskin, and S. Toruńczyk. Energy and Mean-Payoff Games with Imperfect Information. In *CSL '10*, LNCS 6247, pages 260–274, 2010.
10 J. Fearnley. Exponential Lower Bounds for Policy Iteration. In *ICALP '10*, volume 6199 of *LNCS*, pages 551–562. Springer, 2010.
11 W. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39:162–175, 1991.
12 C. Lusena, J. Goldsmith, and M. Mundhenk. Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14, 2001.
13 O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.
14 K. Murphy. A Survey of POMDP Solution Techniques. Technical report, 2000.
15 D. Thorsley and D. Teneketzis. Active acquisition of information for diagnosis and supervisory control of discrete event systems. *Discrete Event Dynamic Systems*, 17(4):531–583, 2007.
16 M. Tracol. Recurrence and transience for finite probabilistic tables. *Theoretical Computer Science*, 412(12-14):1154–1168, 2011.
17 M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS '85*, pages 327–338. IEEE, 1985.

Optimal Packed String Matching *

Oren Ben-Kiki[1], Philip Bille[2], Dany Breslauer[3], Leszek Gąsieniec[4], Roberto Grossi[5], and Oren Weimann[6]

1 **Intel Research and Development Center, Haifa, Israel**
2 **Technical University of Denmark, Copenhagen, Denmark**
3 **Caesarea Rothchild Institute, University of Haifa, Haifa, Israel**
4 **University of Liverpool, Liverpool, United Kindom**
5 **Dipartimento di Informatica, Università di Pisa, Pisa, Italy**
6 **University of Haifa, Haifa, Israel**

Abstract

In the packed string matching problem, each machine word accommodates α characters, thus an n-character text occupies n/α memory words. We extend the Crochemore-Perrin constant-space $O(n)$-time string matching algorithm to run in optimal $O(n/\alpha)$ time and even in real-time, achieving a factor α speedup over traditional algorithms that examine each character individually. Our solution can be efficiently implemented, unlike prior theoretical packed string matching work. We adapt the standard RAM model and only use its AC^0 instructions (i.e., no multiplication) plus two specialized AC^0 packed string instructions. The main *string-matching* instruction is available in commodity processors (i.e., Intel's SSE4.2 and AVX Advanced String Operations); the other *maximal-suffix* instruction is only required during pattern preprocessing. In the absence of these two specialized instructions, we propose theoretically-efficient emulation using integer multiplication (not AC^0) and table lookup.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, F.2.2 Nonnumerical Algorithms and Problems—Pattern Matching

Keywords and phrases String matching, Bit parallelism, Real time, Space efficiency

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.423

1 Introduction

Hundreds of articles have been published about string matching, exploring the multitude of theoretical and practical facets of this fundamental problem. For an n-character text T and an m-character pattern x, the classical algorithm by Knuth, Morris and Pratt [21] takes $O(n+m)$ time and uses $O(m)$ auxiliary space to find all pattern occurrences in the text, namely, all text positions i, such that $T[i..i+m-1] = x$. Many other algorithms have been published; some are faster on the average, use only constant auxiliary space, operate in real-time, or have other interesting benefits. In an extensive study, Faro and Lecroq [12] offer an experimental comparative evaluation of some 85 string matching algorithms.

Packed strings. In modern computers, the size of a machine word is typically larger than the size of an alphabet character and the machine level instructions operate on whole words, i.e., 64-bit or longer words vs. 8-bit ASCII, 16-bit UCS, 2-bits biological DNA, 5-bits amino acid alphabets, etc. The packed string representation fits multiple characters into one

* Partially supported by the European Research Council (ERC) project SFEROT, by the Israeli Science Foundation grant 347/09 and by Italian project PRIN AlgoDEEP (2008TFBWL4) of MIUR.

larger word, so that the characters can be compared in bulk rather than individually: if the characters of a string are drawn from an alphabet Σ, then a word of $\omega \geq \log_2 n$ bits fits up to α characters, where the packing factor is $\alpha = \frac{\omega}{\log_2 |\Sigma|} \geq \log_{|\Sigma|} n$.[1]

Using the packed string representation in the string matching problem is not a new idea and goes back to early string matching papers by Knuth, Morris and Pratt [21, §4] and Boyer and Moore [6, §8-9], to times when hardware character byte addressing was new and often less efficient than word addressing. Since then, several practical solutions that take advantage of the packed representation have been proposed in the literature [2, 4, 11, 15, 16, 25]. However, none of these algorithms improves over the worst-case $O(n)$ time bounds of the traditional algorithms. On the other hand, any string matching algorithm should take at least $\Omega(n/\alpha)$ time to read a packed text in the worst case, so there remains a gap to fill.

Existing work. A significant theoretical step recently taken introduces a few solutions based on either tabulation (a.k.a. "the Four-Russian technique") or word-level parallelism (a.k.a. "bit-parallelism"). Fredriksson [15, 16] used tabulation and obtained an algorithm that uses $O(n^\varepsilon m)$ space and $O(\frac{n}{\log_{|\Sigma|} n} + n^\varepsilon m + occ)$ time, where occ denotes the number of pattern occurrences and $\varepsilon > 0$ denotes an arbitrary small constant. Bille [5] improved these bounds to $O(n^\varepsilon + m)$ space and $O(\frac{n}{\log_{|\Sigma|} n} + m + occ)$ time. Very recently, Belazzougui [3] showed how to use word-level parallelism to obtain $O(m)$ space and $O(\frac{n}{m} + \frac{n}{\alpha} + m + occ)$ time. Belazzougui's algorithm uses a number of succinct data structures as well as hashing: for $\alpha \leq m \leq n/\alpha$, his time bound is optimal while space occupancy is not. As admitted by the above authors, none of these results is practical. A summary of the known bounds and our new result is given in Table 1, where our result uses two instructions described later on.

Table 1 Comparison of packed string matching algorithms.

Time	Space	Reference
$O(\frac{n}{\log_{\lvert\Sigma\rvert} n} + n^\varepsilon m + occ)$	$O(n^\varepsilon m)$	Fredriksson [15, 16]
$O(\frac{n}{\log_{\lvert\Sigma\rvert} n} + m + occ)$	$O(n^\varepsilon + m)$	Bille [5]
$O(\frac{n}{\alpha} + \frac{n}{m} + m + occ)$	$O(m)$	Belazzougui [3]
$O(\frac{n}{\alpha} + \frac{m}{\alpha} + occ)$	$O(1)$	This paper

Our results. We propose an $O(n/\alpha + m/\alpha)$ time string matching algorithm (where the term m/α is kept for comparison with the other results) that is derived from the elegant Crochemore-Perrin [9] algorithm. The latter takes linear time, uses only constant auxiliary space, and can be implemented in real-time following the recent work by Breslauer, Grossi and Mignosi [7] – benefits that are also enjoyed in our settings. The algorithm has an attractive property that it compares the text characters only moving forward on two wavefronts without ever having to back up, relying on the celebrated Critical Factorization Theorem [8, 22].

We use a *specialized word-size packed string matching instruction* to anchor the pattern in the text and continue with bulk character comparisons that match the remainder of the pattern. Our reliance on a specialized packed string matching instruction is not far fetched, given the recent availability of such instructions in commodity processors, which has been a catalyst for our work. Our algorithm is easily adaptable to situations where the packed string matching instruction and the bulk character comparison instruction operate on different word sizes. The output occurrences are compactly provided in a bit-mask that can be spelled

[1] Assume that $|\Sigma|$ is a power of two, ω is divisible by $\log_2 |\Sigma|$, and the packing factor α is a whole integer.

out as an extensive list of text positions in extra $O(occ)$ time.

Unlike the prior theoretical work, our solution has a cache-friendly sequential memory access without using large external tables or succinct data structures, and therefore, can also be efficiently implemented. The same specialized packed string matching instruction could also be used in other string matching algorithms, e.g. the Knuth-Morris-Pratt algorithm [19, §10.3.3], but our algorithm also works in real-time and uses only constant auxiliary space.

Model of computation. We adapt the standard word-RAM model with ω-bit words and with only AC^0 instructions (i.e., arithmetic, bitwise and shift operations but no multiplication) plus two other specialized AC^0 instructions. The main word-size packed string matching instruction is available in the recent *Advanced String Operations* in *Intel's Streaming SIMD Extension (SSE4.2) and Advanced Vector Extension (AVX) Efficient Accelerated String and Text Processing* instruction set [18, 20]. The other instruction, which is only used in the pattern preprocessing, finds the lexicographically maximum suffix. Specifically, adopting the notation $[d] = \{0, 1, \ldots, d-1\}$, the two instructions are the following ones:

Word-Size String Matching (`wssm`): find occurrences of one short pattern x that fits in one word (up to α characters) in a text y that fits in two words (up to $2\alpha - 1$ characters). The output is a binary word Z of $2\alpha - 1$ bits such that its ith bit $Z[i] = 1$ iff $y[i..i + |x| - 1] = x$, for $i \in [2\alpha - 1]$. When $i + |x| - 1 \geq \alpha$, this means that only a prefix of x is matched.

Word-Size Lexicographically Maximum Suffix (`wslm`): given a packed string x that fits in one word (up to α characters), return position $i \in [\alpha]$ such that $x[i..\alpha - 1]$ is lexicographically maximum among the suffixes in $\{x[j..\alpha - 1] \mid j \in [\alpha]\}$.

If these instructions are not available, then we can emulate them, but our proposed emulations cause a small slowdown of $\log\log\omega$ as shown in Table 2.

Table 2 Bounds in the word-RAM when the ω-bit `wssm` and `wslm` instructions are not available.

Time	Space	Reference
$O\left(\omega + \frac{n \log\log\omega}{\alpha} + \frac{m}{\alpha} + occ\right)$	$O(1)$	This paper

2 Packed String Matching

In this section we describe how to solve the *packed string matching* problem using the two specialized word-size string matching instructions `wssm` and `wslm`, and standard word-RAM bulk comparisons of packed strings.

▶ **Theorem 1.** *Packed string matching for a length m pattern and a length n text can be solved in $O(\frac{m}{\alpha} + \frac{n}{\alpha})$ time in the word-RAM extended with constant-time `wssm` and `wslm` instructions. Listing explicitly the occ text positions of the pattern occurrences takes an additional $O(occ)$ time. The algorithm can be made real-time, and uses just $O(1)$ auxiliary words of memory besides the read-only $\frac{m}{\alpha} + \frac{n}{\alpha}$ words that store the input.*

The algorithm behind Theorem 1 follows the classical scheme, in which a text scanning phase is run after the pattern preprocessing. In the following, we first present the necessary background and then describe how to perform the text scanning phase using `wssm`, and the pattern preprocessing using `wslm`.

2.1 Background

Critical Factorization. Properties of periodic strings are often used in efficient string algorithms. A string u is *a period* of a string x if x is a prefix of u^k for some integer k, or equivalently if x is a prefix of ux. The shortest period of x is called *the period* of x and its length is denoted by $\pi(x)$. A *substring* or a *factor* of a string x is a contiguous block of symbols u, such that $x = x'ux''$ for two strings x' and x''. A *factorization* of x is a way to break x into a number of factors. We consider factorizations of a string $x = uv$ into two factors: a *prefix* u and a *suffix* v. Such a factorization can be represented by a single integer and is *non-trivial* if neither of the two factors is equal to the empty string.

Given a factorization $x = uv$, a *local period* of the factorization is defined as a non-empty string p that is consistent with both sides u and v. Namely, (i) p is a suffix of u or u is a suffix of p, and (ii) p is a prefix of v or v is a prefix of p. The shortest local period of a factorization is called *the local period* and its length is denoted by $\mu(u, v)$. A non-trivial factorization $x = uv$ is called a *critical factorization* if the local period of the factorization is of the same length as the period of x, i.e., $\mu(u, v) = \pi(uv)$. See Figure 1.

```
  a | b a a a b a        a b | a a a b a        a b a | a a b a
b a   b a              a a a b   a a a b            a   a
      (a)                    (b)                    (c)
```

Figure 1 The local periods at the first three non-trivial factorizations of the string abaaaba. In some cases the local period overflows on either side; this happens when the local period is longer than either of the two factors. The factorization (b) is a critical factorization with local period aaab of the same length as the global period abaa.

Crochemore-Perrin algorithm. Although critical factorizations may look tricky, they allow for a simplification of the text processing phase of string matching algorithms. We assume that the reader is familiar with the Crochemore-Perrin algorithm [9] and its real-time variation Breslauer-Grossi-Mignosi [7]. Observe that Crochemore and Perrin use Theorem 2 to break up the pattern as $x = uv$ for non-empty prefix u and suffix v, such that $|u| \leq \pi(x)$.

▶ **Theorem 2.** *(Critical Factorization Theorem, Cesari and Vincent [8, 22]) Given any* $|\pi(x)| - 1$ *consecutive non-trivial factorizations of a string* x*, at least one is critical.*

Then, they exploit the critical factorization of $x = uv$ by matching the longest prefix z of v against the current text symbols, and using Theorem 3 whenever a mismatch is found.

▶ **Theorem 3.** *(Crochemore and Perrin [9]) Let* $x = uv$ *be a critical factorization of the pattern and let* p *be any local period at this factorization, such that* $|p| \leq \max(|u|, |v|)$*. Then* $|p|$ *is a multiple of* $\pi(x)$*, the period length of the pattern.*

Precisely, if $z = v$, they show how to declare an occurrence of x. Otherwise, the symbol following z in v is mismatching when compared to the corresponding text symbol, and the pattern x can be *safely* shifted by $|z| + 1$ positions to the right (there are other issues for which we refer the reader to [9]).

To simplify the matter in the rest of the paper, we discuss how to match the pattern suffix v assuming without loss of generality that $|u| \leq |v|$. Indeed, if $|u| > |v|$, the Crochemore-Perrin approach can be simplified as shown in [7]: use two critical factorizations, $x = uv$ and $x' = u'v'$, for a prefix x' of x such that $|x'| > |u|$ and $|u'| \leq |v'|$. In this way, matching both u' and v' suitably displaced by $|x| - |x'|$ positions from matching v, guarantees that x

occurs. This fact enables us to focus on matching v and v', since the cost of matching u' is always dominated by the cost of matching v', and we do not need to match u. For the sake of discussion, it suffices to consider only one instance, namely, suffix v.

We now give more details on the text processing phase, assuming that the pattern preprocessing phase has correctly found the critical factorization of the pattern x and its period $\pi(x)$, and any additional pattern preprocessing that may be required (Section 2.3).

While other algorithms may be used with the `wssm` instruction, the Crochemore-Perrin algorithm is particularly attractive because of its simple text processing. Therefore, it is convenient to assume that the period length and critical factorization are exactly computed in the pattern preprocessing burying the less elegant parts in that phase.

2.2 Text processing

The text processing has complementary parts that handle short patterns and long patterns. A pattern x is *short* if its length is at most α, namely, the packed pattern fits into a single word, and is *long* otherwise. Processing short patterns is immediate with `wssm` and, as we shall see, the search for long patterns reduces to that for short patterns.

Short patterns. When the pattern is already short, `wssm` is repeatedly used to directly find all occurrences of the pattern in the text.

▶ **Lemma 4.** *There exists an algorithm that finds all occurrences of a short pattern of length $m \leq \alpha$ in a text of length n in $O(\frac{n}{\alpha})$ time using $O(1)$ auxiliary space.*

Proof. Consider the packed text blocks of length $\alpha + m - 1$ that start on word boundaries, where each block overlaps the last $m - 1$ characters of the previous block and the last block might be shorter. Each occurrence of the pattern in the text is contained in exactly one such block. Repeatedly use the `wssm` instruction to search for the pattern of length $m \leq \alpha$ in these text blocks whose length is at most $\alpha + m - 1 \leq 2\alpha - 1$. ◀

Long patterns. Let x be a long pattern of length $m > \alpha$: occurrences of the pattern in the text must always be spaced at least the period $\pi(x)$ locations apart. We first consider the easier case where the pattern has a long period, namely $m \geq \pi(x) > \alpha$, and so there is at most one occurrence starting within each word.

▶ **Lemma 5.** *There exists an algorithm that finds all occurrences of a long-period long pattern of length $m \geq \pi(x) \geq \alpha$, in a text of length n in $O(\frac{n}{\alpha})$ time using $O(1)$ auxiliary space.*

Proof. The Crochemore-Perrin algorithm can be naturally implemented using the `wssm` instruction and bulk character comparisons. Given the critical factorization $x = uv$, the algorithm repeatedly searches using `wssm` for an occurrence of a prefix of v of length $\min(|v|, \alpha)$ starting in each packed word aligned with v, until such an occurrence is discovered. If more than one occurrence is found starting within the same word, then by Lemma 3, only the first such occurrence is of interest. The algorithm then uses the occurrence of the prefix of v to anchor the pattern within the text and continues to compare the rest of v with the aligned text and then compares the pattern prefix u, both using bulk comparison of words containing α packed characters. Bulk comparisons are done by comparing words; in case of a mismatch the mismatch position can be identified using bitwise `xor` operation, and then finding the most significant set bit.

A mismatch during the attempt to verify the suffix v allows the algorithm to shift the pattern ahead until v is aligned with the text after the mismatch. A mismatch during the

attempt to verify u, or after successfully matching u, causes the algorithm to shift the pattern ahead by $\pi(x)$ location. In either case the time adds up to only $O(\frac{n}{\alpha})$. ◀

When the period of the pattern is shorter than the word size, that is $\pi(x) \leq \alpha$, there may be several occurrences of the pattern starting within each word. The algorithm is very similar to the long period algorithm above, but with special care to efficiently manipulate the bit-masks representing all the occurrences.

▶ **Lemma 6.** *There exists an algorithm that finds all occurrences of a short-period long pattern of length m, such that $m > \alpha > \pi(x)$, in a text of length n in $O(\frac{n}{\alpha})$ time using $O(1)$ auxiliary space.*

Proof. Let p be the prefix of x of length $\pi(x)$, and write $x = p^r p'$, where p' is a prefix of p. If we can find the maximal runs of consecutive ps inside the text, then it is easy to locate the occurrences of x. To this end, let $k \leq r$ be the maximum positive integer such that $k \cdot \pi(x) \leq \alpha$ while $(k+1) \cdot \pi(x) > \alpha$. Note that there cannot exist two occurrences of p^k that are completely inside the same word.

We examine one word w of the text at a time while maintaining the current run of consecutive ps spanning the text word w' preceding w. We apply `wssm` to p^k and $w'w$, and take the rightmost occurrence of p^k whose matching substring is completely inside $w'w$. We have two cases: either that occurrence exists and is aligned with the current run of ps, and so we extend it, or we close the current run and check whether p' occurs soon after. The latter case arises when there is no such an occurrence of p^k, or it exists but is not aligned with the current run of ps. Once all the maximal runs of consecutive occurrences of ps are found (some of them are terminated by p') for the current word w, we can decide by simple arithmetics whether $x = p^r p'$ occurs on the fly. ◀

Real-time algorithm. As mentioned in Section 2.1, the Crochemore-Perrin algorithm can be implemented in real time using two instances of the basic algorithm with carefully chosen critical factorizations [7]. Since we are following the same scheme here, our algorithm reports the output bit-mask of pattern occurrences ending in each text word in $O(1)$ time after reading the word. Thus, we can obtain a real-time version as claimed in Theorem 1.

2.3 Pattern preprocessing

Given the pattern x, the pattern preprocessing of Crochemore-Perrin produces the period length $\pi(x)$ and a critical factorization $x = uv$ (Section 2.1): for the latter, they show that v is the lexicographically maximum suffix in the pattern under either the regular alphabet order or its inverse order, and use the algorithm by Duval [10]. The pattern preprocessing of Breslauer, Grossi and Mignosi [7] uses Crochemore-Perrin preprocessing, and it also requires to find the prefix x' of x such that $|x'| > |u|$ and its critical factorization $x' = u'v'$ where $|u'| \leq |v'|$. Our pattern preprocessing requires to find the period π' for the first α characters in v (resp., those in v'), along with the longest prefix of v (resp., v') having that period. We thus end up with only the following two problems:

1. Given a string x, find its lexicographically maximum suffix v (under the regular alphabet order or its inverse order).
2. Given a string $x = uv$, find its period $\pi(x)$ and the period of a prefix of v.

When $m = O(\frac{n}{\alpha})$, which is probably the case in many situations, we can simply run the above algorithms in $O(m)$ time to solve the above two problems. We focus here on the case when $m = \Omega(\frac{n}{\alpha})$, for which we need to give a bound of $O(\frac{m}{\alpha})$ time.

▶ **Lemma 7.** *Given a string x of length m, its lexicographically maximum suffix v can be found in $O(\frac{m}{\alpha})$ time.*

Proof. Duval's algorithm [10] is an elegant and simple linear-time algorithm that can be easily adapted to find the lexicographically maximum suffix. It maintains two positions i and j, one for the currently best suffix and the other for the current candidate. Whenever there is a mismatch after matching k characters ($x[i+k] \neq x[j+k]$), one position is "defeated" and the next candidate is taken. Its implementation in word-RAM is quite straightforward, by comparing α characters at a time, except when the interval $[\min(i,j), \max(i,j)+k]$ contains less than α positions, and so everything stays in a single word: in this case, we can potentially perform $O(\alpha)$ operations for the $O(\alpha)$ characters (contrarily to the rest, where we perform $O(1)$ operations). We show how to deal with this situation in $O(1)$ time. We employ `wslm`, and let w be the suffix thus identified in the word. We set i to the position of w in the original string x, and j to the first occurrence of w in x *after* position i (using `wssm`). If j does not exist, we return i as the position of the lexicographically maximum suffix; otherwise, we set $k = |w|$ and continue by preserving the invariant of Duval's algorithm. ◀

▶ **Lemma 8.** *The preprocessing of a pattern of length m takes $O(\frac{m}{\alpha})$ time.*

3 Word-Size Instruction Emulation

Our algorithm uses two specialized word-size packed string matching instructions, `wssm` and `wslm`, that are assumed to take $O(1)$ time. In the circuit complexity sense both are AC^0 instructions, which are easier than integer multiplication that is not AC^0, since integer multiplication can be used to compute the parity [17]. Recall that the class AC^0 consist of problems that admit polynomial size circuits of depth $O(1)$, with Boolean `and`/`or` gates of unbounded fan-in and `not` gates only at the inputs.

While either instruction can be emulated using the four Russians' technique, table lookup limits the packing factor and has limited practical value for two reasons: it sacrifices the constant auxiliary space and has no more cache friendly access. We focus here on the easier and more useful main instruction `wssm` and propose efficient bit parallel emulations in the word-RAM, relying on integer multiplication for fast Boolean convolutions.

▶ **Lemma 9.** *After a preprocessing of $O(\omega)$ time, the $\omega/\log\log W$-bit* `wssm` *and* `wslm` *instructions can be emulated in $O(1)$ time on a ω-bit word RAM.*

3.1 Bit-parallel emulation of `wssm`

String matching problems under *general matching relations* were classified in [23, 24] into easy and hard problems, where easy problems are equivalent to string matching and are solvable in $O(n+m)$ time, and hard problems are at least as hard as one or more Boolean convolutions, that are solved using FFT and integer convolutions in $O(n \log m)$ time [1, 14]. To efficiently emulate the `wssm` instruction we introduce *two layers* of increased complexity: first, we observe that the problem can also be solved using Boolean convolutions, and then, we use the powerful, yet standard, integer multiplication operation, that resembles integer convolutions, to emulate Boolean convolutions. In the circuit complexity sense Boolean convolution is AC^0, and therefore, is easier than integer multiplication.

String matching and bitwise convolution via integer multiplication. Consider the Boolean vectors $t_0 \cdots t_{n-1}$ and $p_0 \cdots p_{m-1}$: we need to identify those positions k, such that $t_{k+i} = p_i$, for all $i \in [m]$. Given a text and a pattern, where each of their characters is

encoded in $\log_2|\Sigma|$ bits, we can see them as Boolean vectors of length $\log_2|\Sigma|$ times the original one. We can therefore focus on binary text and pattern. We want to compute the occurrence vector c, such that c_k indicates if there is a pattern occurrence starting at text position $k \in [n]$ (so we then have to select only those c_k that are on $\log_2|\Sigma|$ bit boundaries in c). In general, we have

$$c_k = \bigwedge_{i=0,\ldots,m-1} (t_{k+i} = p_i) = \overline{\left(\bigvee_{i=0,\ldots,m-1} (t_{k+i} \wedge \overline{p_i})\right) \vee \left(\bigvee_{i=0,\ldots,m-1} (\overline{t_{k+i}} \wedge p_i)\right)}.$$

Define the *OR-AND Boolean convolution operator* $\hat{c} = a \bigtriangledown b$ for the Boolean vectors $a = a_{n-1}\cdots a_0$, $b = b_{m-1}\cdots b_0$, and $\hat{c} = \hat{c}_{n+m-1}\cdots\hat{c}_0$, to be

$$\hat{c}_k = \bigvee_{i=\max\{0,k-(n-1)\},\ldots,\min\{m-1,k\}} (a_{k-i} \wedge b_{m-i-1}).$$

Then, the occurrence vector c can be computed by taking the least n significant bits from the outcome of two convolutions, $\hat{c} = \overline{(t \bigtriangledown \overline{p}) \vee (\overline{t} \bigtriangledown p)}$. Treating the Boolean vectors as binary integers with the left shift operator $\ll$, we can compute $a \bigtriangledown b$ using standard integer multiplication $a \times b$, but the sum has to be replaced by the OR operation:

$$a \bigtriangledown b = \bigvee_{i=0,\ldots,m-1} [(a \ll i) \times b_i] = a \times b \quad \text{(where } + \text{ is replaced by } \vee\text{)}.$$

Observe the following to actually use the plain standard integer multiplication $a \times b$. Since the sum of up to m Boolean values is at most m, it can be represented by $L = \lceil \log m + 1 \rceil$ bits. If we pad each digit of a and b with L zeros, and think of each group of $L+1$ bits as a *field*, by adding up at most m numbers the fields would not overflow. Thus, performing the integer multiplication on the padded a and b gives fields with zero or non-zero values (where each field actually counts the number of mismatches). Adding the two convolutions together we get the overall number of mismatches, and we need to identify the fields with no mismatches, corresponding to occurrences and compact them. In other words, if we use padded vectors $t', \overline{t'}, p'$, and $\overline{p'}$, we can compute $r = (t' \times \overline{p'}) + (\overline{t'} \times p')$ and set $\hat{c}_k = 0$ if and only if the the corresponding field in r is non-zero.

We use the constant time word-RAM bit techniques in Fich [13] to pad and compact. Note that in each field with value f we have that $0 - f$ is either 0, or borrows from the next field 1s on the left side. Take a mask with 1 in each field at the least significant bit, and subtract our integer m from this mask. We get that only zero fields have 0 in their most significant bit. Boolean AND with the mask to keep the most significant bit in each field, then shift right to the least significant bit in the field. The only caveat in the above "string matching via integer multiplication" is its need for padding, thus extending the involved vectors by a factor of $L = \Theta(\log m) = O(\log w)$ since they fit into one or two words. We now have to use L machine words, incurs a slowdown of $\Omega(L)$. We next show how to reduce the required padding from L to $\log\log\alpha$.

Sparse convolutions via deterministic samples. A *deterministic sample (DS)* for a pattern with period length π is a collection of at most $\lceil \log \pi \rceil$ pattern positions, such that any two occurrence candidate text locations that match the pattern at the DS must be at least π locations apart [26]. To see that a DS exists, take π consecutive occurrence candidates. Any two candidates must have at least one mismatch position; add one such position to the DS and keep only the remaining minority candidates, removing at least half of the remaining candidates. After at most $\lceil \log \pi \rceil$ iterations, there remains only one candidate and its DS. Moreover, if the input characters are expanded into $\log_2|\Sigma|$ bits, then the DS specifies only

$\lceil \log \pi \rceil$ bits, rather than characters. Candidates can be eliminated via Boolean convolutions with the two bit vectors representing the 0s and 1s in the DS, that is, sparse Boolean vectors with at most $\lceil \log \pi \rceil$ set bits. The period π, the DS, and the other required masks and indices are precomputed in $O(\omega)$ time.

Consider now how we performed string matching via integer multiplication in the previous paragraph. Then, the padding in the bitwise convolution construction can be now reduced to only $L' = \lceil \log \log \pi + 1 \rceil$ bits instead of L bits, leading to convolutions of shorter $O(\omega \log \log \pi) = O(\omega \log \log \omega)$ bit words and slowdown of only $O(\log \log \omega)$ time. Using ω-bit words and $O(\omega)$-time preprocessing, we can treat $O(\omega / \log \log \omega)$ bits in $O(1)$ time using multiplication, thus proving Lemma 9.

3.2 `wssm` on contemporary commodity processors

Benchmarks of packed string matching instructions in "Efficient Accelerated String and Text Processing: Advanced String Operations" *Streaming SIMD Extension (SSE4.2) and Advanced Vector Extension (AVX)* on Intel Sandy Bridge processors [18, 20] and Intel's Optimization Reference Manual [19] indicate remarkable performance. The instruction *Packed Compare Explicit Length Strings Return Mask (PCMPESTRM)* produces a bit mask that is suitable for short patterns and the similar instruction *Packed Compare Explicit Length Strings Return Index (PCMPESTRI)* produces only the index of the first occurrence, which is suitable for our longer pattern algorithm.

Faro and Lecroq kindly made their *String Matching Algorithms Research Tool (SMART)* available [12]. Benchmarks show that for up to 8-character patterns, the raw packed string matching instructions outperformed all existing algorithms in SMART. The Crochemore-Perrin algorithm with packed string matching instructions performed very well on longer patterns. These preliminary experimental results must be interpreted cautiously, since on one hand we have implemented the benchmarks very quickly, while on the other hand the existing SMART algorithms could benefit as well from packed string matching instructions and from other handcrafted machine specific optimization; in fact, a handful of the existing SMART algorithms already use other Streaming SIMD Extension instructions.

4 Conclusions

We demonstrated how to employ string matching instructions to design optimal packed string matching algorithms in the word-RAM, which are fast both in theory and in practice. There is an array of interesting questions that arise from our investigation. *(1)* Compare the performance of our algorithm using the hardware packed string matching instructions to existing implementations (e.g. Faro and Lecroq [12] and platform specific *strstr* in *glibc*). *(2)* Derive Boyer-Moore style algorithms that may be faster on average and skip parts of the text [6, 27] using packed string matching instructions. *(3)* Extend our results to dictionary matching with multiple patterns [3]. *(4)* Improve our emulation towards constant time with ω-bit words and AC^0 operations. *(5)* Find critical factorizations in linear-time using only equality pairwise symbol comparisons: such algorithms could also have applications in our packed string model, possibly eliminating our reliance on the `wslm` instruction.

References

1 A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

2 R. A. Baeza-Yates. Improved string searching. *Softw. Pract. Exper.*, 19(3):257–271, 1989.

3 D. Belazzougui. Worst Case Efficient Single and Multiple String Matching in the RAM Model. In *Proceedings of the 21st International Workshop On Combinatorial Algorithms (IWOCA)*, pages 90–102, 2010.

4 M. Ben-Nissan and S. Tomi Klein. Accelerating Boyer Moore searches on binary texts. In *Proceedings of the 12th International Conference on implementation and Application of Automata (CIAA)*, pages 130–143, 2007.

5 P. Bille. Fast searching in packed strings. *J. Discrete Algorithms*, 9(1):49–56, 2011.

6 R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. of the ACM*, 20:762–772, 1977.

7 Dany Breslauer, Roberto Grossi, and Filippo Mignosi. Simple Real-Time Constant-Space String Matching. In Raffaele Giancarlo and Giovanni Manzini, editors, *CPM*, volume 6661 of *Lecture Notes in Computer Science*, pages 173–183. Springer, 2011.

8 Y. Césari and M. Vincent. Une caractérisation des mots périodiques. *C.R. Acad. Sci. Paris*, 286(A):1175–1177, 1978.

9 M. Crochemore and D. Perrin. Two-way string-matching. *J. ACM*, 38(3):651–675, 1991.

10 J.P. Duval. Factorizing Words over an Ordered Alphabet. *J. Algorithms*, 4:363–381, 1983.

11 S. Faro and T. Lecroq. Efficient pattern matching on binary strings. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2009.

12 S. Faro and T. Lecroq. The exact string matching problem: a comprehensive experimental evaluation report. Technical Report 0810.2390, arXiv, Cornell University Library, 2011. `http://arxiv.org/abs/1012.2547`.

13 F. E. Fich. Constant time operations for words of length w. Technical report, University of Toronto, 1999. `http://www.cs.toronto.edu/~faith/algs.ps`.

14 M.J. Fischer and M.S. Paterson. String matching and other products. In *Complexity of Computation*, pages 113–125. American Mathematical Society, 1974.

15 K. Fredriksson. Faster string matching with super-alphabets. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 44–57, 2002.

16 K. Fredriksson. Shift-or string matching with super-alphabets. *IPL*, 87(4):201–204, 2003.

17 M. L. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

18 Intel. *Intel® SSE4 Programming Reference*. Intel Corporation, 2007.

19 Intel. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel Co., 2011.

20 Intel. *Intel® Advanced Vector Extensions Programming Reference*. Intel Corporation, 2011.

21 D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6:322–350, 1977.

22 M. Lothaire. *Combinatorics on Words*. Addison-Wesley, Reading, MA, U.S.A., 1983.

23 S. Muthukrishnan and K. V. Palem. Non-standard stringology: algorithms and complexity. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 770–779, 1994.

24 S. Muthukrishnan and H. Ramesh. String Matching Under a General Matching Relation. *Inf. Comput.*, 122(1):140–148, 1995.

25 J. Tarhio and H. Peltola. String matching in the DNA alphabet. *Software Practice Experience*, 27:851–861, 1997.

26 U. Vishkin. Deterministic sampling - a new technique for fast pattern matching. *SIAM J. Comput.*, 20(1):22–40, 1990.

27 Andrew Chi-Chih Yao. The complexity of pattern matching for a random string. *SIAM J. Comput.*, 8(3):368–387, 1979.

Dynamic programming in faulty memory hierarchies (cache-obliviously)*

Saverio Caminiti[1], Irene Finocchi[1], Emanuele G. Fusco[1], and Francesco Silvestri[2]

1 **Computer Science Department,** *Sapienza* **University of Rome**
{caminiti, finocchi, fusco}@di.uniroma1.it
2 **Department of Information Engineering, University of Padova**
silvest1@dei.unipd.it

Abstract

Random access memories suffer from transient errors that lead the logical state of some bits to be read differently from how they were last written. Due to technological constraints, caches in the memory hierarchy of modern computer platforms appear to be particularly prone to bit flips. Since algorithms implicitly assume data to be stored in reliable memories, they might easily exhibit unpredictable behaviors even in the presence of a small number of faults. In this paper we investigate the design of dynamic programming algorithms in faulty memory hierarchies. Previous works on resilient algorithms considered a one-level faulty memory model and, with respect to dynamic programming, could address only problems with local dependencies. Our improvement upon these works is two-fold: (1) we significantly extend the class of problems that can be solved resiliently via dynamic programming in the presence of faults, settling challenging non-local problems such as all-pairs shortest paths and matrix multiplication; (2) we investigate the connection between resiliency and cache-efficiency, providing cache-oblivious implementations that incur an (almost) optimal number of cache misses. Our approach yields the first resilient algorithms that can tolerate faults at any level of the memory hierarchy, while maintaining cache-efficiency. All our algorithms are correct with high probability and match the running time and cache misses of their standard non-resilient counterparts while tolerating a large (polynomial) number of faults. Our results also extend to Fast Fourier Transform.

1998 ACM Subject Classification B.8 [Performance and reliability]; F.2 [Analysis of algorithms and problem complexity]; I.2.8 [Dynamic programming].

Keywords and phrases Unreliable memories, fault-tolerant algorithms, dynamic programming, cache-oblivious algorithms, Gaussian elimination paradigm.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.433

1 Introduction

Random access memories suffer from failures that lead the logical state of some bits to be read differently from how they were last written. A recent study has analyzed the memory-error sensitivity of Google's fleet of commodity servers over a period of nearly two years, observing an incidence of errors much higher than previously reported in laboratory conditions [26]. Due to low supply voltage and low critical charge per cell, caches in the memory hierarchy

* This work was supported in part by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2008TFBWL4 national research project AlgoDEEP. The last author was also supported by the University of Padova under Projects STPD08JA32 and CPDA099949/09.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 433–444

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of modern computer platforms appear to be even more prone to bit flips than dynamic random access memories [22], while sophisticated error-correction algorithms are prohibitive for on-chip caches due to tight constraints on die size. The effect of memory errors is an important consideration in system design, especially for long-running and large-scale applications that work on massive data sets. When hardware techniques to detect bit flips are not available, it is important to design algorithms and data structures that are resilient to memory faults without incurring significant space/time penalties.

So far, algorithmic research related to memory faults mainly focused on fault-tolerant sorting networks [25] and on the design of resilient data structures in different (hardly comparable) models [2, 3, 9]. A variety of results have been recently obtained in a faulty RAM model introduced in [19], where an adversary can corrupt at most δ memory cells of a large unreliable memory during the execution of an algorithm. Problems solved resiliently in the faulty RAM model include sorting [17, 19], dictionaries [5, 18], priority queues [23], counting [7], K-d Trees [21], dynamic data structures [15], and local-dependency dynamic programming [8]. All these works focus on a one-level faulty memory: the only exception is [6], which investigates the connection between fault tolerance and I/O-efficiency in the external memory model [1], addressing resilient dictionaries, priority queues, and sorting. We remark that external memory (and, similarly, cache-aware) algorithms often crucially depend on the knowledge of hardware parameters, such as block or cache line size, and may not adapt well to different memory hierarchies. Cache-oblivious algorithms [20] overcome this issue: they are designed in a two-level ideal-cache model with no explicit dependencies on hardware parameters, and can therefore adapt simultaneously to all levels of the memory hierarchy (see [16] for a survey). To the best of our knowledge, no work but [6] proposes algorithms that are both resilient and cache-efficient. These two requirements pose indeed conflicting challenges: resilient algorithms typically use data replication, which might result in poor spatial locality, and perform error detection and recovery strategies throughout the computation, which might result in poor temporal locality.

Hierarchical faulty memory model. To analyze the resiliency of algorithms to faults that take place anywhere in the memory hierarchy, we combine the notions of fault-tolerance and cache-obliviousness extending the faulty RAM model [19] and the faulty external memory model [6] in a natural way. We assume the existence of a multilevel unreliable memory. Overall, at most δ (adversarial) corruptions can take place: each fault can be inserted at any time during the execution of an algorithm and at any level of the memory hierarchy. The algorithms can exploit knowledge of δ, which is a parameter of the model, but are oblivious to hardware parameters of the memory hierarchy and require neither error detection capabilities nor cryptographic assumptions. Following [20], we analyze the cache complexity in a two-level ideal-cache model, where both levels may be faulty: a fully associative cache of size M is partitioned into lines, each consisting of B consecutive words which are always moved together to/from main memory according to an optimal off-line replacement strategy (these choices are justified in [20]). Similarly to previous works [3, 6, 17, 18], we assume the existence of P private memory words that are incorruptible and hidden from the adversary: the private memory can be used, e.g., in the case of randomized algorithms to store random values and their derivatives. If $P = \Theta(1)$, the private memory can be implemented by a constant number of dedicated registers and accessed without incurring cache misses. If P is not constant (e.g., $P = \Theta(\log n)$, where n is the input size), we assume the existence of a private memory hierarchy whose largest level has size P: at each hierarchy level, private and public (unreliable) memory have the same cache line size.

Our results. We investigate the design of dynamic programming algorithms in the hierarchical faulty memory model. Previous work on resilient dynamic programming (in short, DP) [8] only applies to *local dependency DP problems*, where updates to entries in the DP table are determined by the contents of $O(1)$ neighboring cells: this class of problems includes, e.g., longest common subsequence and certain kinds of sequence alignment, but excludes many practically relevant problems such as Floyd-Warshall all-pairs shortest paths. Furthermore, algorithms in [8] are designed in a one-level memory model and are not cache-efficient. The contribution of this paper is two-fold. As a first result, we remove the local dependency assumption, significantly extending, w.r.t. [8], the class of problems that can be solved resiliently via dynamic programming in the presence of faults. Hinging upon a recursive framework introduced in [10, 11], we design resilient algorithms for all problems that can be solved by triply-nested loops of the type that occur in the standard Gaussian elimination algorithm, most notably all-pairs shortest paths and matrix multiplication. Similar results also apply to the Fast Fourier Transform. We remark that even checking the correctness of dynamic programming computations for non-local problems has been regarded as an elusive goal for many years. All our algorithms are correct with high probability, are parametric in the private memory size, and can tolerate a polynomial number of faults while still matching the running time of their non-resilient counterparts. As a second contribution, our approach yields the first resilient and cache-oblivious algorithms that can tolerate faults at any level of the memory hierarchy, while incurring an (almost) optimal number of cache misses. To obtain our results we introduce some novel techniques which might be of independent interest in the design of resilient algorithms for different problems.

To exemplify our bounds, consider a classical local-dependency DP problem, i.e., computing a longest common subsequence (LCS) of two sequences of length m and n, with $m \geq n$. We solve LCS resiliently and cache-obliviously in $O(nm + \delta n^{c/P} mP)$ time and $O(nm/(MB) + \delta n^{c/P} mP/B)$ cache misses, where M is the unreliable cache size, P is the private memory size (bounded by $O(\log n)$), B is the number of words in a cache line, δ is an upper bound on the number of faults, and $c < P$ is a small constant. Notice that $\Omega(nm/(MB) + \delta m/B)$ is a lower bound on the number of cache misses in the hierarchical faulty memory model [11], and that $n^{c/P} = \Theta(1)$ when $P = \Theta(\log n)$. Our algorithm matches the $\Theta(mn)$ running time of its non-resilient counterpart as long as $\delta = O(n^{1-c/P}/P)$, offering a full spectrum of tradeoffs between private memory size and number of faults. For instance, when $P = \Theta(\log n)$, we can tolerate up to $\delta = O(n/\log n)$ faults and incur a number of cache misses that is either optimal, if δ is also bounded by $O(n/(M \log n))$, or at most a factor of $\log n$ away from optimal. Even when $P = \Theta(1)$, the algorithm can still tolerate a polynomial number of faults within the same bounds of its non-resilient counterpart. Note that the resilient LCS algorithm from [8] incurs $\Theta(nm/B)$ cache misses, even without faults.

Paper organization. After some preliminaries, in Section 3 we introduce the main tools that will allow us to achieve resiliency and cache-efficiency simultaneously: this section is intended as an overview of our techniques, while the algorithms are detailed in Section 4 (which focuses on local-dependency DP problems) and in Section 5 (devoted to the extension to non-local problems). Due to lack of space, proofs of several results and some detailed description are omitted and will appear in the full version of the paper.

2 Preliminaries

Recursive dynamic programming [10, 11]. Our approach hinges upon a recursive framework for dynamic programming, introduced in [10, 11], that we briefly describe here

for completeness. We refer to [10, 11] for a detailed description and analysis. Let X and Y be two sequences of length n and m, respectively (w.l.o.g., let $m \geq n$). As an example we consider the LCS problem, whose standard DP solution is based on the following recurrence:

$$\ell[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \ell[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ \max\{\ell[i,j-1], \ell[i-1,j]\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases} \tag{1}$$

where $\ell[i,j]$ is the length of a longest common subsequence of prefixes $\langle x_1, \ldots, x_i \rangle$ and $\langle y_1, \ldots, y_j \rangle$. Values ℓ can be stored in a DP table C of size $(n+1) \times (m+1)$, and a longest common subsequence of X and Y can be obtained by computing a traceback path starting from entry $C[n,m]$, according to Equation 1.

Let $C[i,j][h,k]$ be the subtable of the dynamic programming table C ranging from row i to row j and from column h to column k. Let vectors $L = C[i-1,j][h-1,h-1]$, $R = C[i,j][k,k]$, $T = C[i-1,i-1][h-1,k]$, and $D = C[j,j][h,k]$ be the *left*, *right*, *top*, and *down boundaries* of the subtable, respectively. Moreover, let $\langle x_i, \ldots, x_j \rangle$ and $\langle y_h, \ldots, y_k \rangle$ be the *projections* of the input sequences X and Y on $C[i,j][h,k]$. The algorithm presented in [10, 11] is implemented by two recursive functions, Boundary and Traceback-Path, that use a divide-and-conquer strategy, logically splitting table C into four quadrants. Boundary performs a forward computation by recursively solving four subproblems: it returns the output boundaries (R and D) of a quadrant, starting from the projections of X and Y on the quadrant and the input boundaries (L and T). Traceback-Path finds the traceback path π through the DP table C by recursively finding the fragments of the path through the quadrants it traverses: given the entry point of path π on the output boundaries of a quadrant, Traceback-Path calls function Boundary to compute the input boundaries of (at most three) subquadrants, and recursively calls itself to compute the fragments of π traversing the subquadrants. In [12] the authors also provide a multicore version of the algorithm: this extension exploits a tiling sequence, depending on the recursion depth, that determines a subdivision of the DP table into a (not necessarily constant) number of quadrants.

Resilient variables [18]. An *r-resilient variable* x consists of $2r+1$ copies of a standard variable. A *reliable write* operation on x means assigning the same value to each copy. Similarly, a *reliable read* means calculating the majority value, which can be done in $\Theta(r)$ time and $O(1)$ space [4], incurring $O(r/B+1)$ cache misses. The majority value is guaranteed to be correct if $r \geq \delta$, since at most δ copies can be corrupted. If $r < \delta$, an r-resilient variable can be corrupted by the adversary, but at the cost of at least $r+1$ faults.

Karp-Rabin fingerprints [24]. Given a vector $A = \langle a_0, \ldots, a_k \rangle$ and a prime number p, a Karp-Rabin fingerprint can be defined as $\varphi_A = \sum_{i=0}^{k} a_i 2^{w(k-i)} \mod p$, where w is the memory word size. Fingerprint φ_A can be incrementally computed in $O(k)$ time and $O(1)$ private memory: when a new number a_i is revealed, φ_A can be updated in $O(1)$ time using Horner's rule and simple modular arithmetics [24].

3 Overview of our techniques

In this section we describe the main tools that will allow us to adapt some cache-oblivious algorithms to run in the presence of memory faults, while keeping the number of cache misses bounded. In Section 4 we will show how to combine these tools to make functions Boundary and Traceback-Path resilient.

Read and write fingerprints. The hierarchical faulty memory model does not provide fault detection capabilities: hence, we need to guarantee that values read throughout the

computation (from the input sequences and from the DP table) were not tampered since they were last written. To this aim, we use read and write Karp-Rabin fingerprints. Since Boundary and Traceback-Path are recursive, we will associate fingerprints to the input and output data of each call (i.e., to quadrant boundaries and to sequence projections). We use an independently generated prime number p_d for each recursion depth d and denote the write and read fingerprints of a vector A as φ_A and $\overline{\varphi}_A$, respectively. The correctness of data stored in A can be checked by reading A, computing $\overline{\varphi}_A$, and comparing its value against the write fingerprint φ_A produced when A was previously written: if $\varphi_A \neq \overline{\varphi}_A$, a fault occurred.

Bounding private data. We store fingerprints, primes, and information about recursive calls in the private memory, whose amount is limited to P memory words. Since $\Omega(1)$ data are necessary per recursion level, we need to limit the depth of the recursion tree, depending on P. Let c be the number of local variables used by the algorithm and let ρ be the largest integer such that $c\,\rho \leq P$: notice that $\rho = \Theta(P)$. At each call, we split the table into $\lambda \times \lambda$ quadrants, where $\lambda = \lceil n^{1/\rho} \rceil$: this guarantees that private data fits into P memory words.

Lazy fault detection. With non-constant λ and $O(1)$ fingerprints per recursion level, checking the correctness of the input for all the recursive calls would result in non-negligible time overhead. Hence, we detect faults lazily and perform fingerprint tests only when all subproblems have been recursively solved.

Data replication at decreasing resiliency levels. To resume computation after the detection of a fault, we use r-resilient variables. Since working at resiliency level δ throughout the computation would asymptotically increase the running time, we exploit a hierarchy of *decreasing resiliency levels*, tied with the depth of the recursive call: calls that are deeper in the recursion tree correspond to smaller subproblems and have a lower level of resiliency. It is worth noticing that the corruption of r-resilient variables, with $r < \delta$, together with lazy fault detection, might force the algorithm to perform entire subtree computations on wrong data. The corruption will be detected only by fingerprints at level r, but the wasted computation time will be amortized on $\Theta(r)$ faults. This will be crucial to bound the total cost of error recovery.

Amplified fingerprints. We will see in Section 4 that, during the execution of the algorithm, read and write data access patterns do not necessarily coincide. This is an issue, since updating a fingerprint $\overline{\varphi}_A$ while reading a vector $A = \langle a_0, \ldots, a_k \rangle$ according to an arbitrary pattern could require logarithmic time per access, due to exponentiation (see Section 2). Moreover, some values are possibly read $\omega(1)$ times and should appear in fingerprints tied with different exponents. To address these issues, we exploit regularities in data access patterns. We define an *amplified write fingerprint* as $\varphi_A = \sum_{i=0}^{k}(a_i \sum_{j=1}^{s_i} 2^{w f_{i,j}}) \mod p_d$, where s_i is the amplifying factor for element a_i (i.e., the number of times a_i will be read), and values $f_{i,j}$ are distinct positive integers characterizing the access pattern. The correctness of read data can be verified by updating the *amplified read fingerprint* $\overline{\varphi}_A$ adding $a_i 2^{w f_{i,j}}$ during the j-th reading of a_i. In our algorithm, factors $2^{w f_{i,j}}$ can be computed in $O(1)$ amortized time: these computations depend on the access pattern and will be therefore discussed later.

4 Recursive local-dependency dynamic programming

We now describe in more details how functions Boundary and Traceback-Path can be made resilient. We present a cache-efficient implementation in Section 4.1 and its analysis in Section 4.2. We assume that the input sequences are δ-resilient and that both of them have length n: the latter assumption can be removed by splitting the longer sequence Y into $\lceil m/n \rceil$ segments.

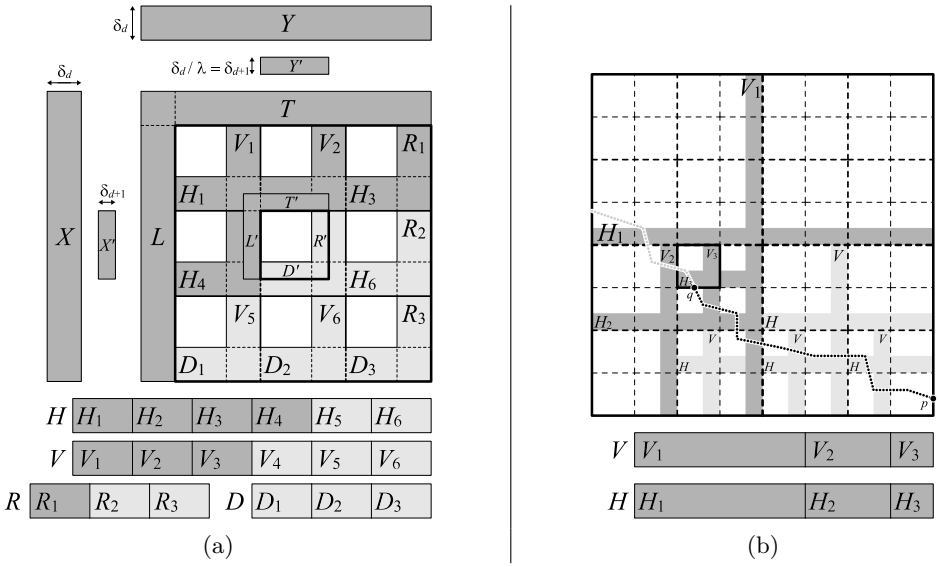


Figure 1 (a) DP quadrants, boundaries, and auxiliary vectors; (b) cache-oblivious traceback path computation with virtual quadrants.

Insert and extract. Throughout the computation, we repeatedly extract and merge vector segments, changing their resiliency level and updating their fingerprints, by means of two auxiliary functions, called `insert` and `extract`. Function `insert` combines two vectors: given as input a vector A stored at resiliency level r, a vector A' stored at resiliency level $r' \leq r$, two write fingerprints φ_A and $\varphi_{A'}$, it reads by majority values in A' and appends them to A, increasing their resiliency from r' to r (we assume that enough memory has been already allocated in A). At the same time, `insert` updates the write fingerprint φ_A with the new values and computes a read fingerprint $\overline{\varphi}_{A'}$ to check correctness of the read data: if $\overline{\varphi}_{A'} \neq \varphi_{A'}$, the function fails. Symmetrically, function `extract` takes a small vector out of a larger one.

Resilient boundary. Function Boundary, when called at recursion depth d, receives as input L, T, and the projections of X and Y, all stored at resiliency level $\delta_d = \lceil \delta/\lambda^d \rceil$, and the corresponding write fingerprints produced by its caller (initialization can be appropriately done at recursion depth 0). All the input vectors have the same length $n_d = \lceil n/\lambda^d \rceil$ and are stored in the unreliable memory, while fingerprints are private. Similarly to [12], Boundary recursively solves λ^2 subproblems in row-major order. However, the parameters of the recursive calls are vectors of length $n_{d+1} = \lceil n/\lambda^{d+1} \rceil$ stored at resiliency level $\delta_{d+1} = \lceil \delta/\lambda^{d+1} \rceil$, together with their write fingerprints, and the output is stored into two auxiliary (horizontal and vertical) vectors H and V. These vectors have length $(\lambda - 1)n_d$, resiliency δ_d, and are associated with a write and a read fingerprint (see Fig. 1.a). Appropriate fingerprint tests are also performed during the computation, as detailed below. Consider an internal quadrant $\langle i, j \rangle$, with $1 < i, j < \lambda$. During the recursive call at depth d, the algorithm works as follows:

Step 1. The input boundaries L' and T', of length n_{d+1} and resiliency δ_{d+1}, are extracted from V and H, respectively, together with their write fingerprints. The projection Y' of sequence Y on the quadrant is obtained similarly (X' is extracted from X only if $j = 1$). These `extract` operations also update the read fingerprints $\overline{\varphi}_H$, $\overline{\varphi}_V$, and $\overline{\varphi}_Y$.

Step 2. Given L', T', Y', and X' (together with their write fingerprints) computed in step

1, the recursive call on quadrant $\langle i, j\rangle$ returns the output boundaries D' and R', of length n_{d+1} and resiliency δ_{d+1}, and their write fingerprints $\varphi_{D'}$ and $\varphi_{R'}$. If this call fails, all data at resiliency level δ_{d+1} are discarded, the prime number p_{d+1} associated with recursion level $d+1$ is renewed, and the computation restarts from step 1. Backup copies of $\overline{\varphi}_V$, $\overline{\varphi}_H$, and $\overline{\varphi}_Y$ are used to restore the computation state. The projection of X is extracted once per row and requires an additional backup fingerprint based on prime number p_d.

Step 3. Upon successful termination of the recursive call, the output boundaries D' and R' of quadrant $\langle i, j\rangle$ are merged with H and V, respectively. If `insert` fails (due to a fingerprint mismatch on δ_{d+1}-resilient data) the computation is restarted from step 1. Otherwise, the write fingerprints φ_H and φ_V are updated.

Quadrants on the first column and quadrants on the first row (i.e., $j = 1$ and $i = 1$, respectively) are handled similarly, but L' and T' are extracted from L and T (instead of V and H), respectively. When computing quadrants on the last row or column, the resulting output is inserted into the output vectors R and D with resiliency δ_d.

No fingerprint test is performed at the end of step 1 to establish the correctness of the extracted subsequences: such a test would require reading V and H from scratch, and would have a prohibitive running time. The following fingerprint tests are instead (lazily) performed, depending on the quadrant:

- $\overline{\varphi}_V = \varphi_V$ and $\overline{\varphi}_H = \varphi_H$ must hold if $\langle i, j\rangle = \langle \lambda, \lambda\rangle$;
- $\overline{\varphi}_L = \varphi_L$ and $\overline{\varphi}_T = \varphi_T$ must hold if $\langle i, j\rangle = \langle \lambda, 1\rangle$ or $\langle 1, \lambda\rangle$, respectively;
- $\overline{\varphi}_Y = \varphi_Y$ must hold for each $i \in [1, \lambda]$ and $j = \lambda$;
- $\overline{\varphi}_X = \varphi_X$ must hold if $\langle i, j\rangle = \langle \lambda, 1\rangle$.

A mismatch on any of the above tests implies failure of the recursive call at depth d and will be handled by higher recursive calls (see step 2).

Resilient traceback path. The resilient implementation of Traceback-Path computes the traceback path segment π traversing a quadrant, stored at resiliency level δ_d, and its write fingerprint φ_π. Traceback-Path calls resilient Boundary to obtain vectors H and V, containing the output boundaries (at resiliency level δ_d) of the λ^2 quadrants of size $n_{d+1} \times n_{d+1}$. Then, it computes π backward from H and V by calling itself on (at most $2\lambda - 1$) subquadrants intersected by π. Segments of π (at resiliency level δ_{d+1}) obtained by the recursive calls are stitched and increased in resiliency using function `insert`. Fingerprint mismatches at resiliency level δ_d cause the current call of Traceback-Path to fail, while mismatches at level δ_{d+1} or failed subroutine invocations cause the computation to be repeated. Since the backward access pattern to H, V, L, and T is inverted with respect to the order in which data are written, we use amplified fingerprints as described in Section 3. E.g., the read fingerprint of vector H can be efficiently computed by saving in the private memory a running value $2^{w(|H|-i-1)} \mod p_d$ and performing a single multiplication by 2^w per update. We also notice that some quadrants may not be intersected by the traceback path, but we force the algorithm to read vector segments corresponding to these quadrants in order to correctly update the read fingerprints.

4.1 Cache-oblivious implementation

To improve temporal locality we access data in Z-order [20], and to improve spatial locality we shrink the size of data structures in the unreliable memory by recycling space as soon as written data are no longer needed. While this is quite standard in the design of cache-oblivious algorithms, it has non-trivial consequences on fingerprint computation.

Amplified fingerprints vs. Z-order. When using the Z-order, read operations on vectors H and V do not follow the write Z-order in which their write fingerprints φ_H and φ_V have been produced. We have thus to change the read fingerprint computation to reflect this different order while maintaining $O(1)$ amortized time per operation. Consider, e.g., the computation of $\overline{\varphi}_H$ for calls at recursion depth $d = \rho - 1$ (similar reasonings can be applied to the other vectors and recursion depths). Since $d = \rho - 1$, we have that $n_d \leq \lambda$, each subproblem is a single entry of the DP table C, and vector H corresponds to a portion of C of size $n_d \times n_d$. We use $\varphi_H = \sum_{x=0}^{|H|-1} H[x]2^{wx} \mod p_d$ as a write fingerprint. The computation of cell (i, j), requires values from cells $(i, j-1)$, $(i-1, j)$, and $(i-1, j-1)$. It can be shown that the ranks of these cells in vector H (say r_1, r_2, and r_3) can be computed from the rank r of cell (i, j) as $r_1 = r - \text{left}(\exp(j))$, $r_2 = r - \text{up}(\exp(i))$, and $r_3 = r - \text{diag}(\exp(i), \exp(j))$, where $\exp(k)$ is the exponent of 2 in the factorization of k, $\text{diag}(h, k) = \text{up}(h) + \text{left}(k)$, and up and left are defined as follows:

$$\text{left}(k) = \begin{cases} 1 & \text{if } k = 0 \\ 2^{2k-1} + \text{left}(k-1) & \text{otherwise} \end{cases} \qquad \text{up}(k) = \begin{cases} 2 & \text{if } k = 0 \\ 2^{2k} + \text{up}(k-1) & \text{otherwise} \end{cases}$$

Since each written value is read three times (from below, from the right, and from the bottom-right diagonal), we maintain three read fingerprints $\overline{\varphi}_{H,R}$, $\overline{\varphi}_{H,B}$, and $\overline{\varphi}_{H,D}$. While reading, e.g., cell $(i, j-1)$ to compute cell (i, j), we update $\overline{\varphi}_{H,R}$ by adding $H[r_1]2^{w(|H|-r)} \cdot 2^{-w(r-r_1)} \mod p_d$. To make this computation efficient, we precompute the inverse 2^{-w} of 2^w in the ring $\mathcal{Z}_{p_d}$, for all selected primes p_d. Computing $\text{left}(\exp(j))$ for all $j \in [0, \lambda - 1]$ requires $O(\lambda)$ sums and divisions by 2, and thus the amortized cost of each computation is constant. Within the same bound it is also possible to compute $2^{-w \,\text{left}(\exp(j))} \mod p_d$, starting from 2^{-w} and performing constantly many products for each sum in the computation of $\text{left}(\exp(j))$. Similar reasonings apply to functions up and diag.

Amplified fingerprints vs. virtual quadrants. To optimize cache misses, the length of vectors H and V, at recursion depth d, should be reduced from λn_d to $\Theta(n_d)$. We adapt a technique proposed in [11, 12]: at any time, only appropriate subvectors of H and V are stored, obtaining the missing parts, when necessary, by repeating forward computations. In more details, the $(n_d \times n_d)$-size DP table is split into four quadrants of size $(n_d/2 \times n_d/2)$, which are superimposed over the $\lambda \times \lambda$ submatrices. Let Q be the quadrant containing the entry point of π: the input boundaries for Q are obtained by applying function Boundary to at most three quadrants, and the traceback path π is computed by combining the output of at most three recursive calls of Traceback-Path on the intersected quadrants. It can be shown that the additional forward computations do not asymptotically increase the running time, and that the active boundaries stored in H and V (see Fig. 1.b) have length $O(n_d)$ [11, 12].

To apply this technique in our setting, two main issues need to be settled. A first technical issue is that recursion on $(n_d/2^i \times n_d/2^i)$-size quadrants cannot be explicit, since the recursion tree would exceed the amount of private memory when $P = o(\log n)$. Hence, in our implementation $(n_d/2^i \times n_d/2^i)$-size quadrants are only *virtual*: recursive calls of Traceback-Path on virtual quadrants inside an $n_d \times n_d$ quadrant are simulated iteratively (this can be done using only a constant number of indexes and variables), while real recursive calls are performed when $n_d/2^i$ becomes smaller than n_{d+1}. The resiliency is kept at level δ_d during simulated recursion, and drops to δ_{d+1} on real recursive calls.

The main issue is that, with virtual quadrants, the data access pattern becomes more complex and requires opportunely crafted amplified fingerprints to enable error detection while keeping negligible the time overhead. Consider as an example vector H and suppose for the moment that recursive calls of function Traceback-Path are performed, during simulated

recursion, on all quadrants. The access pattern on H is given by a layer of $\log_2 \lambda$ levels, where layer i corresponds to an inverted Z-order on 4^i data segments of length $n_d/2^i$. Hence, the number of elements in layer i is $n_d 2^i$ and the overall number of elements preceding the first element of layer i is given by $\sum_{j=1}^{i-1} n_d 2^i = n_d(2^i - 2)$. Write fingerprints are computed according to the subdivision in layers. Let l be the current depth of simulated recursion and let D' be the output received from a call to function Boundary. Layers involved in the fingerprint computation are layers l to $\log_2 \lambda$. As elements $h \in D'$ are inserted into H, φ_H is updated in constant amortized time by adding $h \sum_{i=l}^{\log_2 \lambda} 2^{w(n_d(2^i-2)+r_i n_d/2^i+s_i)}$ mod p_d, where r_i is the rank, in layer i, of the segment containing h and s_i is the number of elements preceding h in this segment. Vector V and the projections of the input sequences are handled in a similar way. Updates to read fingerprints are done accordingly to the current layer on quadrants intersected by the traceback path π, paying attention to work also on quadrants that are not intersected by π (to keep the read fingerprints consistent with the write fingerprints).

4.2 Analysis

Let α be the number of faults actually introduced by the adversary during an execution of the algorithm: notice that $\alpha \leq \delta$. We recall that the two input sequences have length m and n, with $m \geq n$, and that $\rho = \Theta(\log_\lambda n) = \Theta(P)$ is the recursion depth (see Section 3).

▶ **Theorem 1.** *Algorithm* Resilient-LCS *computes, with high probability, a correct longest common subsequence of the input sequences X and Y.*

Proof. If no memory fault is introduced by the adversary, the correctness of the algorithm follows from [10]. In general, the first call of function Traceback-Path has resiliency $\delta_0 = \delta$: this implies that majority values cannot be corrupted and computation is never aborted. It is not difficult to see that at most $\rho + \alpha$ selections of prime numbers are needed during an execution of the algorithm. It follows from [8] that, for any constants k and γ, it is possible to independently select $\rho + \alpha$ numbers in $[m^{k-1}, m^k]$, uniformly at random, so that all selected numbers are prime with probability at least $1 - (\rho + \alpha)/m^\gamma$. We now consider the probability that fingerprint tests do not fail. It can be shown using standard techniques that each fingerprint test fails to identify a corrupted variable with probability at most $1/(\sigma m^{k-2})$, for some positive constant σ. Since no more than α variables can be corrupted by the adversary during the execution of the algorithm, we have that the overall probability of detecting all faults is at least $1 - \alpha/(\sigma m^{k-2})$, provided that all selected numbers are primes. The probability that the algorithm computes the LCS correctly is thus at least $(1 - \alpha/(\sigma m^{k-2}))(1 - (\rho + \alpha)/m^\gamma)$. By appropriately choosing constants k and γ, this probability can be made larger than $1 - 1/m^\varepsilon$, for any $\varepsilon > 0$. ◀

The following theorem gives the running time of algorithm Resilient-LCS.

▶ **Theorem 2.** *Algorithm* Resilient-LCS *requires $O(mn + \delta m n^{c/P} P)$ time in the worst case, where P is the available private memory, $c < P$ is a small constant, m and n (with $m \geq n$) are the lengths of the input sequences, and δ is an upper bound on the number of memory faults.*

Proof. Algorithm Resilient-LCS consists of $\lceil m/n \rceil$ calls of Traceback-Path with input size n. Hence, functions Boundary and Traceback-Path are always called on two strings of length n. We first consider the time spent in successful computation and then take into account the time spent in computation discarded due to the detection of some fault. W.l.o.g

we assume n, δ and λ to be powers of two. Since at some recursion depth d the resiliency level $\delta_d = \delta/\lambda^d$ may become smaller than one, we suppose vectors to be $\Theta(\delta_d + 1)$-resilient.

Consider a successful computation of function Boundary at recursion depth d with input size n_d and resiliency level δ_d. If $n_d \leq \lambda$, the function requires $T_B(n_d, \delta_d) = O\left(n_d^2(\delta_d + 1)\right)$ time. If $n_d > \lambda$, the time $T_B(n_d, \delta_d)$ becomes $O\left(n_d^2 + \delta_d n_d \lambda \log_\lambda n_d\right)$ since Boundary performs λ^2 recursive calls with input size n_d/λ and resiliency δ_d/λ, and each call requires $O\left(n_d(\delta_d + 1)/\lambda\right)$ time for preparing inputs and fingerprints. Therefore $T_B(n, \delta) = O(n^2 + \delta n \lambda \log_\lambda n) = O(n^2 + \delta n n^{c/P} P)$, by definition of λ.

Inducing a recomputation at level $1 \leq i \leq k$, with $k = \log_\lambda \min\{n, \delta\}$, requires δ/λ^i faults (there cannot be recomputation at level $i = 0$ since boundaries are δ-resilient). Hence at most $\alpha\lambda^i/\delta$ recomputations can be induced. Since there are λ^{2i} subproblems at level i, the following summation bounds from above the time spent in unsuccessful computation:

$$\sum_{i=1}^{k} \frac{\alpha\lambda^i}{\delta} \frac{T_B(n, \delta)}{\lambda^{2i}} \leq T_B(n, \delta) \sum_{i=1}^{k} \frac{1}{\lambda^i} \leq T_B(n, \delta)$$

If $\delta < n$, recursive calls done at levels deeper than k are all done at resiliency level 1. The adversary can induce up to α recomputations at these levels, each of which has cost bounded by $T_B(n, \delta)/\lambda^{2k}$. Hence the time spent in unsuccessful computation at levels $j \in [k + 1, \log_\lambda n]$ is upper bounded by: $\alpha T_B(n, \delta)/\lambda^{2k} = \alpha T_B(n, \delta)/\delta^2 < T_B(n, \delta)$. In all cases, this time does not exceed the time spent in successful computation.

Consider a successful computation of Traceback-Path at recursion depth d with input size n_d and resiliency level δ_d. If $n_d \leq \lambda$ the function requires $O\left(n_d^2(\delta_d + 1)\right)$ time. If $n_d > \lambda$, the time is $O\left(n_d^2 + n_d\delta_d\lambda^{\log_2 3} \log_\lambda n_d\right)$ since the function performs at most $2\lambda - 1$ recursive calls with input size n_d/λ and resiliency δ_d/λ, and calls at most 3^j times function Boundary with input size $n_d/2^j$ and resiliency $\delta_d + 1$, for each $1 \leq j \leq \log_2 \lambda$. As done previously, it can be shown that the time spent in unsuccessful computation does not exceed the time spent in successful computation. Multiplying these bounds by $\lceil m/n \rceil$ calls and recalling that $\lambda = n^{1/\Theta(P)}$, the theorem follows. ◀

Theorem 2 implies that, when $P = \Theta(\log n)$ and $\delta = O(n/\log n)$, the running time of algorithm Resilient-LCS is $O(nm)$, matching the running time of the non resilient cache-oblivious algorithm given in [10]. Furthermore, for any small constant private memory, the algorithm can still tolerate a polynomial number of faults within the non-resilient bounds. Theorem 3 gives the cache complexity of algorithm Resilient-LCS.

▶ **Theorem 3.** *Algorithm* Resilient-LCS *incurs* $O(mn/(BM) + \delta mn^{c/P}P/B)$ *cache misses in the worst case, where* $c < P$ *is a small constant.*

It follows from Theorem 3 that, when the available private memory is $\Theta(\log n)$ and $\delta = O(n/(M \log n))$, algorithm Resilient-LCS incurs $O(nm/(BM))$ misses, which is optimal for algorithms based on Equation (1), as proved in [10]. Notice that, in the hierarchical faulty memory model, we have an additional lower bound $\Omega(\delta m/B)$, given by the number of cache misses needed to read the input sequences at resiliency level δ. Hence, in any case, the number of cache misses is at most a $\log n$ factor away from optimal.

5 Extension to non-local problems

The techniques described in previous sections can be used to extend significantly the class of problems that are efficiently solvable in the presence of memory faults. Here, we sketch

resilient and cache-efficient algorithms for problems that fit in the Gaussian Elimination Paradigm [11] and for the Fast Fourier Transform. These algorithms compute the correct solution with high probability, when up to δ faults are inserted by an adversary.

Gaussian Elimination Paradigm (GEP). This paradigm includes all problems that can be solved by a triply nested `for` loop which updates each entry of an $n \times n$ input matrix C at most n times. Some notable examples are matrix multiplication, Gaussian elimination and LU decomposition without pivoting, and Floyd-Warshall all-pairs shortest paths. I-GEP [13] is a subclass of GEP which includes all the aforementioned problems. In I-GEP, it is possible to perform certain reorderings of the updates of matrix C guaranteeing that the final result remains correct, despite the fact that the intermediate states of C are different. In [12, 13, 14], cache-oblivious algorithms for I-GEP are provided for single processors, parallel, and multicore machines. Our algorithm Resilient-I-GEP is based on the multicore version [12]: it works recursively and relies on a subdivision of the input matrix into a varying number of square subproblems, depending on the recursion depth.

Let $\lambda = \lceil n^{1/\rho} \rceil$ be defined as in Section 3, where $\rho = \Theta(P)$. At recursion depth d, Resilient-I-GEP receives as input four δ_d-resilient $n_d \times n_d$ submatrices of C, which are divided into λ^2 submatrices of size $n_d/\lambda \times n_d/\lambda$. Initially, the four matrices coincide with C, which is stored δ-resiliently. The algorithm performs λ passes on these submatrices, solving λ^3 subproblems in total: the four input matrices of each subproblem are stored $\lceil \delta_d/\lambda \rceil$-resiliently. The execution of the λ^3 subproblems follows the order described in [13], which guarantees cache efficiency. For each of the $\Theta(P)$ recursive levels, the algorithm stores in the private memory $O(1)$ fingerprints which are opportunely crafted to reflect the execution order of the λ^3 subproblems and are similar in spirit to the amplified fingerprints used in Resilient-LCS. Algorithm Resilient-I-GEP solves I-GEP problems correctly with high probability: its running time is $O\left(n^3 + n^{2+c/P}\delta P\right)$ and the number of cache misses is $O(n^3/(B\sqrt{M}) + n^{2+c/P}\delta P/B)$, where $c < P$ is a suitable small constant. If $P = \Theta(\log n)$, the algorithm matches the running time of its non-resilient counterpart when $\delta = O(n/\log n)$. Optimal cache efficiency is achieved for $\delta = O(n/(M \log n))$.

Fast Fourier Transform (FFT). Algorithm Resilient-FFT is built on the cache-oblivious FFT algorithm in [20]. It computes the FFT of a δ-resilient n-size vector by computing $2\sqrt{n}$ FFTs on $\lceil \delta/\sqrt{n} \rceil$-resilient $\sqrt{n}$-size vectors. As usual, each input vector is associated with a fingerprint stored in private memory. The algorithm is correct with high probability and requires $\Theta(\log \log n)$ private memory. Its running time is $O(n \log n + n\delta)$, while the cache complexity is $O((n \log_M n)/B + n\delta/B + 1)$. The running time matches the corresponding bound of the non-resilient algorithm when $\delta = O(\log n)$. Optimal cache efficiency is achieved for $\delta = O(\log_M n)$. For larger values of δ, the algorithm matches the resilient lower bounds given by the misses and time required for reading the input vector resiliently.

References

1 A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.

2 Y. Aumann and M. A. Bender. Fault tolerant data structures. In *Proc. 37th FOCS*, pages 580–589, 1996.

3 M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2–3):225–244, 1994.

4 R. S. Boyer and J. S. Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.

5 G. S. Brodal, R. Fagerberg, I. Finocchi, F. Grandoni, G. F. Italiano, A. G. Jørgensen, G. Moruz, and T. Mølhave. Optimal resilient dynamic dictionaries. In *Proc. 15th ESA*, volume 4698 of *LNCS*, pages 347–358, 2007.

6 G. S. Brodal, A. G. Jørgensen, and T. Mølhave. Fault tolerant external memory algorithms. In *Proc. 11th WADS*, volume 5664 of *LNCS*, pages 411–422, 2009.

7 G. S. Brodal, A. G. Jørgensen, G. Moruz, and T. Mølhave. Counting in the presence of memory faults. In *Proc. 20th ISAAC*, volume 5878 of *LNCS*, pages 842–851, 2009.

8 S. Caminiti, I. Finocchi, and E. G. Fusco. Local dependency dynamic programming in the presence of memory faults. In *STACS*, volume 9 of *LIPIcs*, pages 45–56, 2011.

9 V. Chen, E. Grigorescu, and R. de Wolf. Efficient and error-correcting data structures for membership and polynomial evaluation. In *Proc. 27th STACS*, volume 5 of *LIPIcs*, pages 203–214, 2010.

10 R. A. Chowdhury, H. S. Le, and V. Ramachandran. Cache-oblivious dynamic programming for bioinformatics. *Trans. Comput. Biology Bioinform.*, 7(3):495–510, 2010.

11 R. A. Chowdhury and V. Ramachandran. Cache-oblivious dynamic programming. In *Proc. 17th SODA*, pages 591–600, 2006.

12 R. A. Chowdhury and V. Ramachandran. Cache-efficient dynamic programming algorithms for multicores. In *Proc. 20th SPAA*, pages 207–216, 2008.

13 R. A. Chowdhury and V. Ramachandran. The cache-oblivious gaussian elimination paradigm: Theoretical framework, parallelization and experimental evaluation. *Theor. Comput. Syst.*, 47:878–919, 2010.

14 R. A. Chowdhury, F. Silvestri, B. Blakeley, and V. Ramachandran. Oblivious algorithms for multicores and network of processors. In *Proc. 24th IPDPS*, 2010.

15 Paul Christiano, Erik D. Demaine, and Shaunak Kishore. Lossless fault-tolerant data structures with additive overhead. In *Proc. 14th WADS*, volume 6844 of *LNCS*, 2011.

16 E. D. Demaine. Cache-oblivious algorithms and data structures. Lecture Notes from the EEF Summer School on Massive Data Sets, BRICS, 2001.

17 I. Finocchi, F. Grandoni, and G. F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. *Theor. Comput. Sci.*, 410(44):4457–4470, 2009.

18 I. Finocchi, F. Grandoni, and G. F. Italiano. Resilient dictionaries. *ACM Trans. on Algorithms*, 6(1), 2009.

19 I. Finocchi and G. F. Italiano. Sorting and searching in faulty memories. *Algorithmica*, 52(3):309–332, 2008.

20 M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th FOCS*, pages 285–298, 1999.

21 Fabian Gieseke, Gabriel Moruz, and Jan Vahrenhold. Resilient k-d trees: K-means in space revisited. In *ICDM*, pages 815–820, 2010.

22 B. L. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008.

23 A. G. Jørgensen, G. Moruz, and T. Mølhave. Priority queues resilient to memory faults. In *Proc. 10th WADS*, volume 4619 of *LNCS*, pages 127–138, 2007.

24 R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987.

25 F. T. Leighton, Y. Ma, and C. G. Plaxton. Breaking the $\Theta(n \log^2 n)$ barrier for sorting with faults. *J. Comput. Syst. Sci.*, 54(2):265–304, 1997.

26 B. Schroeder, E. Pinheiro, and W. D. Weber. DRAM errors in the wild: a large-scale field study. *Commun. ACM*, 54(2):100–107, 2011.

Deciding Probabilistic Simulation between Probabilistic Pushdown Automata and Finite-State Systems

Hongfei Fu*[1] and Joost-Pieter Katoen†[2]

1,2 RWTH Aachen University, Ahornstraße 55, D-52074 Aachen, Germany

Abstract

This paper studies the decidability and computational complexity of checking probabilistic simulation pre-order between probabilistic pushdown automata (pPDA) and (probabilistic) finite-state systems. We show that checking classical and combined probabilistic similarity are EXPTIME-complete in both directions and become polynomial if both the number of control states of the pPDA and the size of the finite-state system are fixed. These results show that checking probabilistic similarity is as hard as checking similarity in the standard, i.e., non-probabilistic setting.

1998 ACM Subject Classification F.2.2, F.3.1

Keywords and phrases infinite-state systems, probabilistic simulation, probabilistic pushdown automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.445

1 Introduction

Probabilities are a convenient means to model uncertainty [15, 11] in transition systems. They are essential for modelling, e.g., randomized algorithms, unreliable and unpredictable system behaviour, or environmental uncertainties. Discrete-Time Markov Chains (DTMC) and Markov Decision Processes (MDPs) are popular probabilistic extensions of transition systems. An important technique in the formal verification of probabilistic systems is semantical equivalence or pre-order checking. Here, one focuses on comparing the behaviour of a probabilistic system (the "implementation") with its intended behaviour (the "specification"). Probabilistic bisimulation [16, 18, 3] and simulation [16, 12, 18, 3] are typical instances that act as a basis for comparison. For finite-state systems, efficient algorithms have been established for both notions [6, 1, 2]. An additional interest in checking probabilistic (bi)simulation between two systems is the preservation of temporal logic formulas: e.g., probabilistic bisimulation equivalence preserves PCTL while probabilistic simulation equivalence preserves a safety fragment of PCTL [3, 18].

This paper considers probabilistic pushdown automata (pPDA) [7] as implementation models and Segala's probabilistic automata [18] as specifications. pPDA are a natural abstract model for probabilistic procedural programs and are equally expressive as recursive Markov chains [9]. In fact, there are linear-time transformations between these two models [9]. Whereas the verification of pPDA and recursive Markov chains has been addressed quite extensively in the literature [8, 13], their use in semantical equivalence checking has so far been restricted to a study of (strong) probabilistic bisimulation [5]. Probabilistic

* Supported by a CSC scholarship.
† Supported by the EU FP7 project MoVeS.

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 445–456

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

automata [18] are an orthogonal extension of labelled transition systems with probabilities and subsume both DTMCs and MDPs. They are used as semantical model of probabilistic process algebras and constitute the core of PIOA, an input/output version that is frequently used for describing randomized distributed algorithms. In this paper we study the decidability and complexity of checking (combined) strong simulation pre-order between pPDA and finite-state probabilistic automata. Our motivation is that system specifications can often be captured by a finite system, thus for many practical problems it is useful and sufficient to check if a (possibly infinite) probabilistic system is semantically equivalent to a finite one.

Our approach to tackle the above problem is to attempt to lift techniques for related problems in the non-probabilistic setting to the probabilistic one. The obvious candidate is the result by Kučera and Mayr [14] stating that strong simulation pre-order between PDA and finite-state systems (in both directions) is decidable in EXPTIME. Their proof technique relies on the EXPTIME-completeness of model checking modal μ-Calculus against PDA. As model checking PCTL over fully probabilistic PDA is undecidable [4], extending this approach is however hopeless. Instead, we take an alternative route and extend the method by Stirling [19, 20] for showing the decidability of bisimulation equivalence over PDA's. This extension will be non-trivial as (non-probabilistic) strong simulation pre-order is undecidable over PDA's [10]. Our technique enables us to show that probabilistic (combined) simulation pre-order (in both directions) is in EXPTIME. The problem is decidable in PTIME if both the number of control states of the pPDA and the size of the finite-state system are fixed. By exploiting a hardness result in [14], we achieve that the considered problem is EXPTIME-complete.

2 Preliminaries

2.1 Probabilistic Transition Systems

In this subsection we introduce the notion of probabilistic transition systems (pTS) which corresponds to "simple probabilistic automata" defined in [18].

▶ **Definition 1** (Probability Distribution)**.** Let S be a countable set. A function $\mu : S \to [0,1]$ is a *probability distribution over* S if $\sum_{s\in S} \mu(s) = 1$. We say that μ is *finite*, if the set $\lfloor\mu\rfloor := \{s \in S \mid \mu(s) > 0\}$ is finite, and $\mu(s) \in \mathbb{Q}$ for all $s \in S$. Let $\mathcal{D}(S)$ (resp. $\mathcal{D}_{\mathrm{f}}(S)$) denote the set of probability distributions (resp. finite probability distributions) over S.

The definition of probabilistic transition system is given as follows:

▶ **Definition 2** (Probabilistic Transition System)**.** A *probabilistic transition system* (pTS) $\mathcal{T}$ is a triple (S, A, Ω) where S is a countable set of *states*, A is a countable set of *actions* and $\Omega \subseteq S \times A \times \mathcal{D}(S)$ is a set of *transitions*. We define the following notations related to $\mathcal{T}$:

- $\mathrm{der}^{\mathcal{T}}(s,a) := \{\mu \in \mathcal{D}(S) \mid (s,a,\mu) \in \Omega\}$ for $s \in S$ and $a \in A$.
- $\mathrm{Der}^{\mathcal{T}}(s) := \bigcup_{a\in A} \bigcup_{\mu\in \mathrm{der}^{\mathcal{T}}(s,a)} \lfloor\mu\rfloor$ for $s \in S$.
- $\mathrm{Act}^{\mathcal{T}}(s) := \{a \in A \mid \mathrm{der}^{\mathcal{T}}(s,a) \neq \emptyset\}$ for $s \in S$.

We write $s \xrightarrow{a}_{\mathrm{n}} \mu \in \Omega$ instead of $(s,a,\mu) \in \Omega$ (where 'n' stands for "non-combined"). We omit '$\mathcal{T}$', 'Ω' in the notations above if the context is clear.

The following definition illustrates the notion of "combined transitions", which is originally introduced by Segala [18] to model stochastic adversaries.

▶ **Definition 3** (Combined Transitions)**.** Let (S, A, Ω) be a pTS. We write $s \xrightarrow{a}_{\mathrm{c}} \mu$ (where 'c' stands for "combined") if there exists a finite or infinite sequence $\{(\mu_i, d_i)\}_i$ (where

$\mu_i \in \mathcal{D}(S)$ and $d_i \in \mathbb{R}_{\geq 0}$ for every i) such that: (i) $s \xrightarrow{a}_{\mathrm{n}} \mu_i$ for all i; (ii) $\sum_i d_i = 1$; and (iii) $\mu(s) = \sum_i d_i \cdot \mu_i(s)$ for all $s \in S$.

In this paper, we will also concern the notion of "finiteness" in a pTS.

▶ **Definition 4.** A pTS $\mathcal{T}$ is *locally finite* if for all $s \in S$ and $a \in A$, $\mathrm{der}(s,a)$ is a finite subset of $\mathcal{D}_{\mathrm{f}}(S)$. Further $\mathcal{T}$ is *finite* if $\mathcal{T}$ is locally finite and both S and A are finite.

2.2 Probabilistic Simulation

In this subsection we introduce the notion of probabilistic simulations which corresponds to *strong simulation* and *strong probabilistic simulation* defined by Segala [18]. Below we fix a pTS $\mathcal{T} = (S, A, \Omega)$. The following definition originates from [12].

▶ **Definition 5.** [12] Let $\mathcal{R} \subseteq S \times S$. The binary relation $\overline{\mathcal{R}} \subseteq \mathcal{D}(S) \times \mathcal{D}(S)$ is defined as follows: $\mu \overline{\mathcal{R}} \nu$ iff there is a weight function $w : S \times S \to [0,1]$ such that:
- for all $s \in S$, $\sum_{t \in S} w(s,t) = \mu(s)$;
- for all $t \in S$, $\sum_{s \in S} w(s,t) = \nu(t)$;
- for all $(s,t) \in S \times S$, if $w(s,t) > 0$ then $(s,t) \in \mathcal{R}$.

For the sake of simplicity, we write $\mu \mathcal{R} \nu$ instead of $\mu \overline{\mathcal{R}} \nu$.

In this paper, it is somewhat convenient to consider an equivalent definition of Definition 5.

▶ **Definition 6.** Let $\mathcal{R} \subseteq S \times S$. The binary relation $\overline{\mathcal{R}} \subseteq \mathcal{D}(S) \times \mathcal{D}(S)$ is defined as follows: $\mu \overline{\mathcal{R}} \nu$ iff there is a weight function $w : \lfloor \mu \rfloor \times \lfloor \nu \rfloor \to [0,1]$ such that:
- for all $s \in \lfloor \mu \rfloor$, $\sum_{t \in \lfloor \nu \rfloor} w(s,t) = \mu(s)$;
- for all $t \in \lfloor \nu \rfloor$, $\sum_{s \in \lfloor \mu \rfloor} w(s,t) = \nu(t)$;
- for all $(s,t) \in \lfloor \mu \rfloor \times \lfloor \nu \rfloor$, if $w(s,t) > 0$ then $(s,t) \in \mathcal{R}$.

Then the definition of probabilistic simulation is given as follows, where $\sqsubseteq_{\mathrm{n}}$ (resp. $\sqsubseteq_{\mathrm{c}}$) corresponds to *strong simulation* (resp. *strong probabilistic simulation*) in [18], respectively.

▶ **Definition 7** (κ-Simulation)**.** Let $\kappa \in \{\mathrm{n}, \mathrm{c}\}$. A binary relation $\mathcal{R} \subseteq S \times S$ is a *κ-simulation* iff for all $(s,t) \in \mathcal{R}$: (i) $\mathrm{Act}(s) = \mathrm{Act}(t)$ and (ii) whenever $s \xrightarrow{a}_{\mathrm{n}} \mu$ there is $t \xrightarrow{a}_{\kappa} \nu$ such that $\mu \mathcal{R} \nu$. The κ-similarity, denoted $\sqsubseteq_\kappa$, is the union of all κ-simulations.

By definition, one can verify that $\sqsubseteq_\kappa$ is a κ-simulation. Below we define approximants of κ-simulation. We use κ to indicate either 'n' or 'c'.

▶ **Definition 8.** The family $\{\sqsubseteq_\kappa^n\}_{n \in \mathbb{N}_0}$ of approximations of $\sqsubseteq_\kappa$ is inductively defined by:
- $\sqsubseteq_\kappa^0 = \{(s,t) \in S \times S \mid \mathrm{Act}(s) = \mathrm{Act}(t)\}$;
- $(s,t) \in \sqsubseteq_\kappa^{n+1}$ iff $(s,t) \in \sqsubseteq_\kappa^0$ and whenever $s \xrightarrow{a}_{\mathrm{n}} \mu$ there is $t \xrightarrow{a}_{\kappa} \nu$ such that $\mu \sqsubseteq_\kappa^n \nu$.

The following property can be easily proved by induction on n.

▶ **Lemma 9.** *For any $n \in \mathbb{N}_0$, $\sqsubseteq_\kappa^{n+1} \subseteq \sqsubseteq_\kappa^n$ and $\sqsubseteq_\kappa \subseteq \sqsubseteq_\kappa^n$.*

Then the relationship between $\sqsubseteq_\kappa$ and $\{\sqsubseteq_\kappa^n\}_{n \in \mathbb{N}_0}$ is clarified in the following lemma.

▶ **Lemma 10.** *If the underlying pTS $\mathcal{T}$ is locally finite, then $s \sqsubseteq_\kappa t$ iff $s \sqsubseteq_\kappa^n t$ for all $n \in \mathbb{N}_0$.*

Proof. Define $\sqsubseteq_\kappa^\omega := \bigcap_{n \in \mathbb{N}_0} \sqsubseteq_\kappa^n$. We prove that $\sqsubseteq_\kappa^\omega = \sqsubseteq_\kappa$. "$\sqsubseteq_\kappa \subseteq \sqsubseteq_\kappa^\omega$" follows directly from Lemma 9. For "$\sqsubseteq_\kappa^\omega \subseteq \sqsubseteq_\kappa$", we prove that $\sqsubseteq_\kappa^\omega$ is a κ-simulation. Fix any $(s,t) \in \sqsubseteq_\kappa^\omega$ and $a \in A$. Define $\mathcal{R} := \bigcup_{\mu \in \mathrm{der}^{\mathcal{T}}(s,a)} \lfloor \mu \rfloor \times \bigcup_{\nu \in \mathrm{der}^{\mathcal{T}}(t,a)} \lfloor \nu \rfloor$. Then $\mathcal{R}$ is finite as $\mathcal{T}$ is locally finite. Consider any $(s',t') \in \mathcal{R}$. If $(s',t') \notin \sqsubseteq_\kappa^\omega$, then there is a minimal $N(s',t') \in \mathbb{N}_0$ such that

$(s',t') \not\sqsubseteq_\kappa^{N(s',t')}$. Define $N = \max\{N(s',t') \mid (s',t') \in \mathcal{R} \setminus \sqsubseteq_\kappa^\omega\}$ (where $\max \emptyset = 0$). Together with Lemma 9, we have $\mathcal{R} \cap \sqsubseteq_\kappa^N = \mathcal{R} \cap \sqsubseteq_\kappa^\omega$. Since $s \sqsubseteq_\kappa^{N+1} t$, then for any $s \xrightarrow{a}_{\mathrm{n}} \mu$, there is $t \xrightarrow{a}_\kappa \nu$ such that $\mu \sqsubseteq_\kappa^N \nu$. Then $\mu(\sqsubseteq_\kappa^N \cap \mathcal{R})\nu$. Thus $\mu \sqsubseteq_\kappa^\omega \nu$ by $\mathcal{R} \cap \sqsubseteq_\kappa^N = \mathcal{R} \cap \sqsubseteq_\kappa^\omega$. ◀

In this paper we consider κ-similarity between two separate pTS's. This is interpreted in the standard way by taking the disjoint union of the two pTS's.

2.3 Probabilistic Pushdown Automata

In this subsection we extend the fully probabilistic pushdown automata defined by Kučera *et al* [13] to our setting.

▶ **Definition 11** (Probabilistic Pushdown Automata)**.** A *probabilistic pushdown automaton (pPDA)* is a quadruple (Q, Γ, L, Δ) where: Q is a finite set of *control states*; Γ is a finite set of *stack symbols*; L is a finite set of *labels*; $\Delta \subseteq (Q \times \Gamma) \times L \times \mathcal{D}_{\mathrm{f}}(Q \times \Gamma^*)$ is a finite set of *transition rules*. Instead of $(pX, a, \mu) \in \Delta$ (where $p \in Q, X \in \Gamma$) we write $pX \overset{a}{\rightarrowtail} \mu \in \Delta$, and omit '$\Delta$' if it is clear from the context.

Semantics of pPDA Let $P = (Q, \Gamma, L, \Delta)$ be a pPDA. For any $\mu \in \mathcal{D}(Q \times \Gamma^*)$ and $\gamma \in \Gamma^*$, the distribution $\mu_\gamma \in \mathcal{D}(Q \times \Gamma^*)$ is defined as follows: for any $p\alpha \in Q \times \Gamma^*$,

$$\mu_\gamma(p\alpha) = \begin{cases} \mu(p\beta) & \text{if } \alpha = \beta\gamma \text{ for some (unique) } \beta \in \Gamma^* \\ 0 & \text{otherwise} \end{cases}$$

Then the pPDA P induces a locally finite pTS (S, A, Ω) with $S = Q \times \Gamma^*$, $A = L$ and $\Omega = \{pX\gamma \xrightarrow{a}_{\mathrm{n}} \mu_\gamma \mid pX \overset{a}{\rightarrowtail} \mu \in \Delta, \gamma \in \Gamma^*\}$. Here elements of the state set $Q \times \Gamma^*$ are called "configurations". Note that $p\beta\gamma \xrightarrow{a}_\kappa \mu$ with $\beta \in \Gamma^+$ iff $p\beta \xrightarrow{a}_\kappa \mu'$ and $\mu = \mu'_\gamma$ for some μ'.

In this paper, we study the decidability and complexity of the following decision problems: given a configuration $p\alpha$ of a pPDA P and a state f of a finite pTS $\mathcal{F}$, decide whether $p\alpha \sqsubseteq_\kappa f$ and whether $f \sqsubseteq_\kappa p\alpha$, where $\kappa \in \{\mathrm{n}, \mathrm{c}\}$. We prove that both of these problems are EXPTIME-complete.

3 Extended Stack Symbols

In this section we adopt and extend the method by Colin Stirling for PDA's, which is called "extended stack symbols" [19, 20], to our setting. Based on extended stack symbols, Stirling presented a tableaux proof system that decides the bisimilarity between two PDA's. Here we establish extended stack symbols for probabilistic simulation. Then in Section 4 we present a tableaux proof system demonstrating the EXPTIME-decidability of probabilistic simulation between pPDA and a finite pTS. Our extended stack symbols will take a different form from Stirling's.

Below we fix a pPDA $P = (Q, \Gamma, L, \Delta)$ and a finite pTS $\mathcal{F} = (F, A, \Omega)$. For any sets $\mathcal{X}, \mathcal{Y}$, we denote $\Pi[\mathcal{X}, \mathcal{Y}] := \mathcal{X} \times \mathcal{Y} \cup \mathcal{Y} \times \mathcal{X}$.

▶ **Definition 12** (Extended Stack Symbol)**.** An *extended stack symbol* U is a function from Q to 2^F. The set of all extended stack symbols is denoted by $\mathcal{E}$.

Intuitively, an extended stack symbol represents the behaviour of configurations with more symbols on the stack and maps this to corresponding states in F. We now extend pPDA P by setting its stack symbol set to $\Gamma \cup \mathcal{E}$. In the extension of P we do not modify Δ, thus a configuration $pU\alpha$ with $U \in \mathcal{E}$ has no outgoing transitions.

Let us compare Stirling's extended symbols with ours. In [19, 20], an extended stack symbol U is a function that maps a control state p to a configuration $q\alpha \in Q \times \Gamma^*$. Then the semantics is rather straightforward: the extended configuration pU behaves exactly as $q\alpha$. However in our setting, U is much more syntactical: qU can be deemed as the set of all $f \in F$ such that $f \sqsubseteq q\alpha$ (or $q\alpha \sqsubseteq f$, which depends on the context) for some $\alpha \in \Gamma^+$. For example, consider $pX\alpha$ (where $p \in Q$, $X \in \Gamma$) and $f \in F$. If $pX\alpha \sqsubseteq f$ then we expect that $pXU \sqsubseteq f$ where $U(q) := \{g \in F \mid q\alpha \sqsubseteq g\}$, for arbitrary $q \in Q$. Analoguously if $f \sqsubseteq pX\alpha$ then we expect that $f \sqsubseteq pXU$, where $U(q) := \{g \in F \mid g \sqsubseteq q\alpha\}$ for arbitrary $q \in Q$. In this way we reduce $pX\alpha$ to pXU with shorter length, which is the key to make the contruction of our tableaux trees finite. Note that to talk about "$pXU \sqsubseteq f$" or "$f \sqsubseteq pXU$" we need to extend simulation preorders to extended stack symbols, which will be captured by "extended κ-simulations". First we define extended configurations concerned in this paper.

▶ **Definition 13** (Extended Configuration)**.** Define $\mathcal{C} := Q \times (\Gamma^* \cdot (\mathcal{E} + \epsilon))$ and $\mathcal{C}_e := Q \times \mathcal{E}$.

$\mathcal{C}$ is the set of extended configurations of concern: we only consider extended configurations that contain at most one extended stack symbol at the end of the configuration. Note that $\mathcal{C} \backslash \mathcal{C}_e = Q \times (\epsilon + \Gamma^+ \cdot (\mathcal{E} + \epsilon))$ is the set of all configurations where the extended stack symbol (if it occurs) is guarded by a non-empty sequence of (normal) stack symbols.

▶ **Definition 14** (Extended κ-Simulation)**.** Define

$$\mathcal{R}_e := \{(qU, f) \in \mathcal{C}_e \times F \mid f \in U(q)\} \cup \{(f, qU) \in F \times \mathcal{C}_e \mid f \in U(q)\}$$

A relation $\mathcal{R} \subseteq \Pi[\mathcal{C}, F]$ is an *extended κ-simulation* iff for all $(s,t) \in \mathcal{R}$

- if $(s,t) \in \Pi[\mathcal{C}_e, F]$ then $(s,t) \in \mathcal{R}_e$; and
- if $(s,t) \in \Pi[\mathcal{C} \backslash \mathcal{C}_e, F]$ then $\mathrm{Act}(s) = \mathrm{Act}(t)$ and whenever $s \xrightarrow{a}_n \mu$ there is $t \xrightarrow{a}_\kappa \nu$ such that $\mu \mathcal{R} \nu$.

The *extended κ-similarity*, denoted $\preceq_\kappa$, is the union of all extended κ-simulations.

Next we extend simulation approximants to extended stack symbols.

▶ **Definition 15.** The family $\{\preceq_\kappa^n\}_{n \in \mathbb{N}_0}$ is inductively defined as follows:

- $\preceq_\kappa^0 := \{(s,t) \in \Pi[\mathcal{C} \backslash \mathcal{C}_e, F] \mid \mathrm{Act}(s) = \mathrm{Act}(t)\} \cup \mathcal{R}_e$;
- $\preceq_\kappa^{n+1} := \{(s,t) \in \Pi[\mathcal{C} \backslash \mathcal{C}_e, F] \mid \mathrm{Act}(s) = \mathrm{Act}(t)$ and $(\forall s \xrightarrow{a}_n \mu)(\exists t \xrightarrow{a}_\kappa \nu).(\mu \preceq_\kappa^n \nu)\} \cup \mathcal{R}_e$.

By similar proofs for $\sqsubseteq_\kappa$, we have the following two lemmas:

▶ **Lemma 16.** *For any $n \in \mathbb{N}_0$, $\preceq_\kappa^{n+1} \subseteq \preceq_\kappa^n$ and $\preceq_\kappa \subseteq \preceq_\kappa^n$.*

▶ **Lemma 17.** *For any $(s,t) \in \Pi[\mathcal{C}, F]$, $s \preceq_\kappa t$ iff $s \preceq_\kappa^n t$ for all $n \in \mathbb{N}_0$.*

The following theorem clarifies the relation between $\preceq_\kappa$ and $\sqsubseteq_\kappa$.

▶ **Theorem 18.** *For any $(s,t) \in \Pi[Q \times \Gamma^*, F]$, $s \preceq_\kappa t$ iff $s \sqsubseteq_\kappa t$.*

Proof. It can be shown by induction on n that $s \preceq_\kappa^n t$ iff $s \sqsubseteq_\kappa^n t$ for all $n \in \mathbb{N}_0$. Then the result follows from Lemma 10 and Lemma 17. ◀

Theorem 18 allows us to decide $\preceq_\kappa$ instead of $\sqsubseteq_\kappa$.

4 Tableaux Proof System

In this section we demonstrate the EXPTIME-decidability of κ-similarity through a tableaux proof system. Below we fix a pPDA $P = (Q, \Gamma, L, \Delta)$ and a finite pTS $\mathcal{F} = (F, A, \Omega)$. We use p, q to range over Q, X, Y to range over Γ, a to range over $A \cup L$, f, g to range over F,

U to range over $\mathcal{E}$, and α, β, γ to range over $\Gamma^* \cdot (\mathcal{E} + \epsilon)$. We extend P with extended stack symbols as described in Section 3.

Due to Theorem 18, the decision problem for κ-similarity can be reformulated and simplified as follows: Given a pair $(s,t) \in \Pi[Q \times \Gamma, F]$, decide if $s \preceq_\kappa t$. We only consider elements in $Q \times \Gamma$ since for any $pX\alpha \in Q \times \Gamma^*$, we can always add a new control state p_{init} and a new stack symbol X_{init} and augment Δ with the set $\{p_{\text{init}}X_{\text{init}} \xrightarrow{a} \mu_\alpha \mid pX \xrightarrow{a} \mu\}$, so that $p_{\text{init}}X_{\text{init}}$ mimics $pX\alpha$.

The Tableaux System We use tableaux to solve this problem along the lines of [19, 20]. A tableaux is a goal-directed proof system that consists of a set of goals Goals and a set RULE of rules which is essentially a decidable subset of $\mathsf{Goals} \times \mathcal{P}_{\text{f}}(\mathsf{Goals})$, where $\mathcal{P}_{\text{f}}(\mathsf{Goals})$ is the set of finite subsets of Goals. Graphically, a rule $(\mathsf{goal}, \{\mathsf{goal}_1, \ldots, \mathsf{goal}_n\}) \in \mathsf{RULE}$ can be viewed as a proof step:

$$\frac{\mathsf{goal}}{\mathsf{goal}_1 \ldots \mathsf{goal}_n}$$

where goal is what currently is to be proved and $\mathsf{goal}_1 \ldots \mathsf{goal}_n$ are the subgoals what it reduces to. Each rule is backward sound: in this instance if all goal_i are true then so is goal. An application of a rule $(\mathsf{goal}, \{\mathsf{goal}_1, \ldots, \mathsf{goal}_n\})$ to a goal goal is to make all the subgoals $\mathsf{goal}_1 \ldots \mathsf{goal}_n$ children of goal. Then a tableaux tree is a proof tree built from a specified goal (the root of the tree) and repeated application of rules. The leaves of a tableaux tree are divided into *terminal* and *nonterminal* leaves. Terminal leaves are divided into *successful* and *unsuccessful* leaves. A tableaux tree is *successful* iff it is finite and all its leaves are successful.

Below we formulate our tableaux proof system. Given a (ordinary) rooted tree T, we define $V(T)$ to be the set of vertices of T, and $lf(T) \subseteq V(T)$ to be the set of leaves of T.

▶ **Definition 19** (Goals). Define $\mathsf{Goals} := \Pi[\mathcal{C}, F]$. A goal $(p\alpha, f) \in \mathsf{Goals}$ (resp. $(f, p\alpha) \in \mathsf{Goals}$) is also written as $p\alpha \preceq f$ (resp. $f \preceq p\alpha$), which corresponds to the proof analogue of $p\alpha \preceq_\kappa f$ (resp. $f \preceq_\kappa p\alpha$). Below we define the notion of basic successfulness of goals:

A goal $s \preceq t$ is *basically successful* iff:	A goal $s \preceq t$ is *basically unsuccessful* iff:
1. $s \preceq t = pU \preceq f$ such that $f \in U(p)$; or	1. $s \preceq t = pU \preceq f$ such that $f \notin U(p)$; or
2. $s \preceq t = f \preceq pU$ such that $f \in U(p)$; or	2. $s \preceq t = f \preceq pU$ such that $f \notin U(p)$; or
3. $(s,t) \in \Pi[\mathcal{C}\backslash\mathcal{C}_{\text{e}}, F]$ and $\text{Act}(s) = \text{Act}(t) = \emptyset$.	3. $(s,t) \in \Pi[\mathcal{C}\backslash\mathcal{C}_{\text{e}}, F]$ and $\text{Act}(s) \neq \text{Act}(t)$.

▶ **Definition 20** (Proof Tree). A *proof tree* is a pair $(T, \mathcal{L})$ for which T is a (possibly infinite) rooted tree and $\mathcal{L} : V(T) \to \mathsf{Goals}$ is a labeling function. Denote $\mathcal{L}(T)$ be the range of $\mathcal{L}$. A leaf $v \in lf(T)$ is *successful* iff either $\mathcal{L}(v)$ is basically successful, or there is $v' \in V(T)$ such that $v' \neq v$, v' lies on the path from the root of T to v and $\mathcal{L}(v') = \mathcal{L}(v)$. A leaf $v \in lf(T)$ is *unsuccessful* iff $\mathcal{L}(v)$ is basically unsuccessful. A leaf $v \in lf(T)$ is *terminal* if either v is successful or unsuccessful; otherwise it is *non-terminal*.

Note that if v with $\mathcal{L}(v) = (s,t)$ is non-terminal, then $(s,t) \in \Pi[\mathcal{C}\backslash\mathcal{C}_{\text{e}}, F]$ and $\text{Act}(s) = \text{Act}(t) \neq \emptyset$. Below we define rules in our tableaux system. There are two kinds of rules in our tableaux system: UNF (Unfolding) and RED (Reduction).

▶ **Definition 21** (Rules). $\text{UNF}_\kappa \subseteq \mathsf{Goals} \times \mathcal{P}_{\text{f}}(\mathsf{Goals})$ is defined by: $(s \preceq t, \mathcal{G}) \in \text{UNF}_\kappa$ iff

- $s \preceq t \in \Pi[Q \times (\Gamma \cdot (\mathcal{E} + \epsilon)), F]$ and $\text{Act}(s) = \text{Act}(t) \neq \emptyset$;
- $\mathcal{G} \subseteq \text{Der}(s) \times \text{Der}(t)$ and for any $s \xrightarrow{a}_{\text{n}} \mu$, there is $t \xrightarrow{a}_\kappa \nu$ such that $\mu \mathcal{G} \nu$.

$\text{RED}_\kappa \subseteq \mathsf{Goals} \times \mathcal{P}_{\text{f}}(\mathsf{Goals})$ is defined as $\text{RED}^{\text{a}}_\kappa \cup \text{RED}^{\text{b}}_\kappa$, where

- $(s \preceq t, \mathcal{G}) \in \mathrm{RED}^{\mathrm{a}}_{\kappa}$ iff $s \preceq t = pX\alpha \preceq f$ such that
  - $\mathrm{Act}(pX) = \mathrm{Act}(f) \neq \emptyset$ and $\alpha \in \Gamma^{+} \cdot (\mathcal{E} + \epsilon)$; and
  - $\mathcal{G} = \{pXU \preceq f\} \cup \{q\alpha \preceq g \mid q \in Q, g \in U(q)\}$ for some $U \in \mathcal{E}$.
- $(s \preceq t, \mathcal{G}) \in \mathrm{RED}^{\mathrm{b}}_{\kappa}$ iff $s \preceq t = f \preceq pX\alpha$ such that
  - $\mathrm{Act}(f) = \mathrm{Act}(pX) \neq \emptyset$ and $\alpha \in \Gamma^{+} \cdot (\mathcal{E} + \epsilon)$; and
  - $\mathcal{G} = \{f \preceq pXU\} \cup \{g \preceq q\alpha \mid q \in Q, g \in U(q)\}$ for some $U \in \mathcal{E}$.

We denote $\mathrm{RULE}_{\kappa} := \mathrm{UNF}_{\kappa} \cup \mathrm{RED}_{\kappa}$

Intuitively, a rule of UNF_{κ} expands a goal $s \preceq t$ one-step further (cf. Lemma 25) and a rule of RED_{κ} reduce a goal $pX\alpha \preceq f$ (resp. $f \preceq pX\alpha$) to $pXU \preceq f$ (resp. $f \preceq pXU$) together with all information at α encoded in U. Here we use 'a' to indicate the case $s \preceq t \in \mathcal{C} \times F$ and 'b' to indicate the case $s \preceq t \in F \times \mathcal{C}$; we will continue using this in the sequel.

The following definition illustrates the application of rules.

▶ **Definition 22** (Rule Application)**.** Suppose $B = (T, \mathcal{L})$ be a proof tree, $v \in lf(T)$ and $(\mathsf{gl}, \mathcal{G}) \in \mathrm{RULE}_{\kappa}$ such that v is non-terminal and $\mathcal{L}(v) = \mathsf{gl}$. The proof tree $(T', \mathcal{L}')$ *after the application* of $(\mathsf{gl}, \mathcal{G})$ at v, denoted $B_v[\mathsf{gl}, \mathcal{G}]$, is defined as follows: $T' = T \cup \{(v, v') | v' \in V^{\mathrm{new}}\}$ and $\mathcal{L}' = \mathcal{L} \cup \mathcal{L}^{\mathrm{new}}$, where $V^{\mathrm{new}} \cap V(T) = \emptyset$ and $\mathcal{L}^{\mathrm{new}}$ a bijection from V^{new} to $\mathcal{G}$.

Then the set of tableaux trees which is closed under rule application is defined as follows.

▶ **Definition 23** (Tableaux Trees)**.** Let $s \preceq t \in \Pi[\mathcal{C}, F]$. The set of *tableaux trees* rooted at $s \preceq t$, denoted $\mathrm{Tab}[s \preceq t]$, is the smallest set satisfying the following conditions:

- The proof tree with only a single root labeled with $s \preceq t$ belongs to $\mathrm{Tab}[s \preceq t]$.
- If $B = (T, \mathcal{L}) \in \mathrm{Tab}[s \preceq t]$, then $B_v[\mathsf{gl}, \mathcal{G}] \in \mathrm{Tab}[s \preceq t]$ for all $v \in lf(T)$, $(\mathsf{gl}, \mathcal{G}) \in \mathrm{RULE}_{\kappa}$ such that v is non-terminal and $\mathcal{L}(v) = \mathsf{gl}$.
- If there is an infinite sequence $\{(T_n, \mathcal{L}_n)\}_{n \in \mathbb{N}_0}$ with each $(T_n, \mathcal{L}_n) \in \mathrm{Tab}[s \preceq t]$ such that $T_n \subseteq T_{n+1}$ and $\mathcal{L}_n \subseteq \mathcal{L}_{n+1}$ for all n, then $(\bigcup_n T_n, \bigcup_n \mathcal{L}_n) \in \mathrm{Tab}[s \preceq t]$.

The tableaux trees have the following finiteness property which can be proved inductively from Definition 23.

▶ **Lemma 24.** *For each $\alpha \in \Gamma^{*}$, we define* $\mathsf{Suffix}(\alpha) := \{\alpha' \mid \alpha'$ *is a suffix of* $\alpha\}$ *(here ϵ is a suffix of any $\alpha \in \Gamma^{*}$). Let* $\mathsf{Suffix} := \bigcup\{\mathsf{Suffix}(\alpha) \mid \exists q \in Q \exists pX \overset{a}{\rightarrowtail} \mu \in \Delta.(q\alpha \in \lfloor \mu \rfloor)\}$*. Define*

- $\mathcal{G}^{\mathrm{a}}_{*} = \{p\beta\alpha \preceq f \mid \beta \in \Gamma \cup \mathsf{Suffix}, \alpha \in \mathcal{E} \cup \{\epsilon\}, p \in Q, f \in F\}$
- $\mathcal{G}^{\mathrm{b}}_{*} = \{f \preceq p\beta\alpha \mid \beta \in \Gamma \cup \mathsf{Suffix}, \alpha \in \mathcal{E} \cup \{\epsilon\}, p \in Q, f \in F\}$

Then if $s \preceq t \in \mathcal{G}^{\mathrm{a}}_{*}$, then for any $(T, \mathcal{L}) \in \mathrm{Tab}[s \preceq t]$, $\mathcal{L}(T) \subseteq \mathcal{G}^{\mathrm{a}}_{*}$. Analoguously if $s \preceq t \in \mathcal{G}^{\mathrm{b}}_{*}$, then for any $(T, \mathcal{L}) \in \mathrm{Tab}[s \preceq t]$, $\mathcal{L}(T) \subseteq \mathcal{G}^{\mathrm{b}}_{*}$.

Below we prove that rules of UNF_{κ} and RED_{κ} are backward sound.

▶ **Lemma 25.** *Let $(s \preceq t, \mathcal{G}) \in \mathrm{UNF}_{\kappa}$. If $s' \preceq^{n}_{\kappa} t'$ for all $s' \preceq t' \in \mathcal{G}$, then $s \preceq^{n+1}_{\kappa} t$.*

Proof. Directly from Definition 15 and Definition 21. ◀

▶ **Lemma 26.** *Let $(pX\alpha \preceq f, \{pXU \preceq f\} \cup \mathcal{G}^{\mathrm{a}}_{\alpha,U}) \in \mathrm{RED}^{\mathrm{a}}_{\kappa}$ where*

$$\mathcal{G}^{\mathrm{a}}_{\alpha,U} := \{q\alpha \preceq g \mid q \in Q, g \in U(q)\}.$$

If $pXU \preceq^{n+1}_{\kappa} f$ and $q\alpha \preceq^{n}_{\kappa} g$ for all $q\alpha \preceq g \in \mathcal{G}^{\mathrm{a}}_{\alpha,U}$, then $pX\alpha \preceq^{n+1}_{\kappa} f$.

Proof. We prove by induction on n that for all $p\gamma\alpha \preceq f \in \mathsf{Goals}$ with $\gamma \in \Gamma^+$ it holds that for any $U \in \mathcal{E}$, if $p\gamma U \preceq_\kappa^{n+1} f$ and $q\alpha \preceq_\kappa^n g$ for all $q\alpha \preceq g \in \mathcal{G}^{\mathrm{a}}_{\alpha,U}$, then $p\gamma\alpha \preceq_\kappa^{n+1} f$.

Base Step: $n = 0$. Since $p\gamma U \preceq_\kappa^1 f$, for any $p\gamma \xrightarrow{a}_{\mathrm{n}} \mu$, there is $f \xrightarrow{a}_\kappa \nu$ such that $\mu_U \preceq_\kappa^0 \nu$. Let $w : \lfloor \mu_U \rfloor \times \lfloor \nu \rfloor \to [0,1]$ be a weight function for $\mu_U \preceq_\kappa^0 \nu$ (cf. Definition 6). We define a weight function $w' : \lfloor \mu_\alpha \rfloor \times \lfloor \nu \rfloor \to [0,1]$ by: $w'(q\beta\alpha, g) = w(q\beta U, g)$ for all $q\beta \in \lfloor \mu \rfloor$ (note that $\beta \in \Gamma^*$) and $g \in \lfloor \nu \rfloor$. We prove that w' is a weight function for $\mu_\alpha \preceq_\kappa^0 \nu$. The first two conditions in Definition 6 are straightforward to verify. For the third condition, suppose $w'(q\beta\alpha, g) > 0$ with $q\beta \in \lfloor \mu \rfloor$. Then $w(q\beta U, g) > 0$ and hence $q\beta U \preceq_\kappa^0 g$. If $\beta \neq \epsilon$ then by Definition 15 $\mathrm{Act}(q\beta) = \mathrm{Act}(g)$ and we have $q\beta\alpha \preceq_\kappa^0 g$. If $\beta = \epsilon$ then $g \in U(q)$ and we have $q\alpha \preceq g \in \mathcal{G}^{\mathrm{a}}_{\alpha,U}$; thus $q\alpha \preceq_\kappa^0 g$. In either case $q\beta\alpha \preceq_\kappa^0 g$. So w' is a weight function for $\mu_\alpha \preceq_\kappa^0 \nu$. Also from $p\gamma U \preceq_\kappa^1 f$ we have $\mathrm{Act}(p\gamma\alpha) = \mathrm{Act}(f)$. So $p\gamma\alpha \preceq_\kappa^1 f$.

Inductive Step: Suppose $p\gamma U \preceq_\kappa^{n+2} f$ and $q\alpha \preceq_\kappa^{n+1} g$ for all $q\alpha \preceq g \in \mathcal{G}^{\mathrm{a}}_{\alpha,U}$. We prove that $p\gamma\alpha \preceq_\kappa^{n+2} f$. Since $p\gamma U \preceq_\kappa^{n+2} f$, for any $p\gamma \xrightarrow{a}_{\mathrm{n}} \mu$, there is $f \xrightarrow{a}_\kappa \nu$ such that $\mu_U \preceq_\kappa^{n+1} \nu$. Consider any $(q\beta U, g) \in \preceq_\kappa^{n+1}$ with $\beta \in \Gamma^*$: if $\beta = \epsilon$ then $q\beta\alpha \preceq_\kappa^{n+1} g$ since $q\alpha \preceq g \in \mathcal{G}^{\mathrm{a}}_{\alpha,U}$; if $\beta \in \Gamma^+$ then we have $q\beta\alpha \preceq_\kappa^{n+1} g$ by induction hypothesis. So by the same construction of weight function in the base step we have $\mu_\alpha \preceq_\kappa^{n+1} \nu$. Also we have $\mathrm{Act}(p\gamma\alpha) = \mathrm{Act}(f)$. Thus $p\gamma\alpha \preceq_\kappa^{n+2} f$. ◀

Similarly we can prove the following analogue to Lemma 26.

▶ **Lemma 27.** *Let* $(f \preceq pX\alpha, \{f \preceq pXU\} \cup \mathcal{G}^{\mathrm{b}}_{\alpha,U}) \in \mathrm{RED}^{\mathrm{b}}_\kappa$ *where*

$$\mathcal{G}^{\mathrm{b}}_{\alpha,U} := \{g \preceq q\alpha \mid q \in Q, g \in U(q)\}.$$

If $f \preceq_\kappa^{n+1} pXU$ *and* $g \preceq_\kappa^n q\alpha$ *for all* $g \preceq q\alpha \in \mathcal{G}^{\mathrm{b}}_{\alpha,U}$, *then* $f \preceq_\kappa^{n+1} pX\alpha$.

Based on Lemma 25 through Lemma 27, we obtain the soundness of our tableaux system:

▶ **Proposition 28.** *If there is a successful tableaux tree rooted at* $s \preceq t$, *then* $s \preceq_\kappa t$.

Proof. We only prove the case when $s \preceq t \in \mathcal{C} \times F$, the other case is similar. The proof is by contraposition. Let $p_0\alpha_0 \preceq f_0 = s \preceq t$. Suppose $(T, \mathcal{L})$ is a successful tableaux tree rooted at $p_0\alpha_0 \preceq f_0$ however $p_0\alpha_0 \not\preceq_\kappa f_0$. Then by Lemma 17, there is $n_0 \in \mathbb{N}_0$ such that $p_0\alpha_0 \preceq_\kappa^{n_0} f_0$ however $p_0\alpha_0 \not\preceq_\kappa^{n_0+1} f_0$. Note that we have $(p_0\alpha_0, f_0) \in \preceq_\kappa^0$ or otherwise the goal $p_0\alpha_0 \preceq f_0$ would be unsuccessful. By the backward soundness of UNF_κ and RED_κ, if the rule applied to $p_0\alpha_0 \preceq f_0$ belongs to UNF_κ, then there is a child $p_1\alpha_1 \preceq f_1$ of $p_0\alpha_0 \preceq f_0$ such that $p_1\alpha_1 \not\preceq_\kappa^{n_0} f_1$; and if the rule applied to $p_0\alpha_0 \preceq f_0$ belongs to $\mathrm{RED}^{\mathrm{a}}_\kappa$, then either $pXU \not\preceq_\kappa^{n_0+1} f$ or $p'\alpha \not\preceq_\kappa^{n_0} f'$ for some $p'\alpha \preceq f' \in \mathcal{G}^{\mathrm{a}}_{\alpha,U}$ with corresponding X, U and α. In either case there is a child $p_1\alpha_1 \preceq f_1$ of $p_0\alpha_0 \preceq f_0$ such that $p_1\alpha_1 \not\preceq_\kappa^{n_0+1} f_1$. Let $n_1 \in \mathbb{N}_0$ such that $p_1\alpha_1 \not\preceq_\kappa^{n_1+1} f_1$ and $p_1\alpha_1 \preceq_\kappa^{n_1} f_1$. Then $n_1 \leq n_0$. In this way we can recursively construct a finite sequence $\{(p_i\alpha_i \preceq f_i, n_i)\}_{1 \leq i \leq k}$ such that $p_i\alpha_i \not\preceq_\kappa^{n_i+1} f_i$ and $p_i\alpha_i \preceq_\kappa^{n_i} f_i$, and the last goal $p_k\alpha_k \preceq f_k$ is a successful leaf. Since $p_k\alpha_k \not\preceq_\kappa^{n_k+1} f_k$, $p_k\alpha_k \preceq_\kappa f_k$ cannot be basically successful. So the only possibility is that there is $j < k$ such that $p_k\alpha_k \preceq f_k = p_j\alpha_j \preceq f_j$. Consider the step from $p_{k-1}\alpha_{k-1} \preceq f_{k-1}$ to $p_k\alpha_k \preceq f_k$:

- if the step is due to UNF_κ, then $n_k < n_{k-1} \leq n_j$;
- if the step is due to $\mathrm{RED}^{\mathrm{a}}_\kappa$ and $p_k\alpha_k \preceq f_k \in \mathcal{G}^{\mathrm{a}}_{\alpha,U}$ with corresponding α and U, then $n_k < n_{k-1} \leq n_j$;
- if the step is due to $\mathrm{RED}^{\mathrm{a}}_\kappa$ and $p_k\alpha_k \preceq f_k = pXU \preceq f$ with corresponding X and U, then $p_{k-1}\alpha_{k-1} \preceq f_{k-1} \neq pXU \preceq f$ and so $j < k-1$. Then from j to $j+1$ the rule is a UNF_κ which implies $n_{j+1} < n_j$, hence $n_k < n_j$.

Thus in either case $n_k < n_j$. But then we have $p_k\alpha_k \not\preceq_\kappa^{n_k+1} f_k$ and $p_k\alpha_k \preceq_\kappa^{n_j} f_k$. Contradiction. ◀

To show the completeness of the tableaux, we first prove a useful lemma below.

▶ **Lemma 29.** *Suppose $pX\alpha \preceq_\kappa f$ and U be an extended symbol such that $U(q) := \{g \in F \mid q\alpha \preceq_\kappa g\}$ for all $q \in Q$. Then $pXU \preceq_\kappa f$.*

Proof. We prove that $\mathcal{R} := \{(q\beta U, g) \mid q \in Q, \beta \in \Gamma^*, q\beta\alpha \preceq_\kappa g\}$ is an extended κ-simulation. Consider any $(q\beta U, g) \in \mathcal{R}$. If $\beta = \epsilon$, then $q\alpha \preceq_\kappa g$ and thus $g \in U(q)$; then $(qU, g) \in \mathcal{R}_{\mathrm{e}}$. On the other hand, suppose that $\beta \in \Gamma^+$. Then $\mathrm{Act}(q\beta U) = \mathrm{Act}(q\beta\alpha) = \mathrm{Act}(g)$. Further for any $q\beta \xrightarrow{a}_{\mathrm{n}} \mu$, by $q\beta\alpha \preceq_\kappa g$ there is $g \xrightarrow{a}_\kappa \nu$ such that $\mu_\alpha \preceq_\kappa \nu$. We prove that $\mu_U \mathcal{R} \nu$. By $\mu_\alpha \preceq_\kappa \nu$, there is a weight function $w : \lfloor \mu_\alpha \rfloor \times \lfloor \nu \rfloor \to [0,1]$ for μ_α and ν. We construct a function $w' : \lfloor \mu_U \rfloor \times \lfloor \nu \rfloor \to [0,1]$ by: $w'(q'\gamma U, g') = w(q'\gamma\alpha, g')$ for all $q'\gamma \in \lfloor \mu \rfloor$ and $g' \in \lfloor \nu \rfloor$ (note that $\gamma \in \Gamma^*$). Then we show that w' is a weight function for μ_U and ν. The first two conditions in Definition 6 are straightforward to verify. For the third condition, consider any $q'\gamma \in \lfloor \mu \rfloor$ and $g' \in \lfloor \nu \rfloor$: if $w'(q'\gamma U, g') > 0$, then $w(q'\gamma\alpha, g') > 0$ and hence $q'\gamma\alpha \preceq_\kappa g'$; then by definition we obtain that $(q'\gamma U, g') \in \mathcal{R}$. Thus $\mathcal{R}$ is an extended κ-simulation. ◀

Similarly, we can obtain the following lemma:

▶ **Lemma 30.** *Suppose $f \preceq_\kappa pX\alpha$ and U be an extended symbol such that $U(q) := \{g \in F \mid g \preceq_\kappa q\alpha\}$ for all $q \in Q$. Then $f \preceq_\kappa pXU$.*

Then the completeness of the tableaux system is as follows:

▶ **Proposition 31.** *Let $s \preceq t \in \mathcal{G}_*^{\mathrm{a}} \cup \mathcal{G}_*^{\mathrm{b}}$. If $s \preceq_\kappa t$ then there is a successful tableaux tree rooted at $s \preceq t$.*

Proof. We only prove the case when $s \preceq t \in \mathcal{G}_*^{\mathrm{a}}$, the other case is similar. Define

$$\mathrm{Tab}'[s \preceq t] := \{(T, \mathcal{L}) \in \mathrm{Tab}[s \preceq t] \mid T \text{ is finite and } s' \preceq_\kappa t' \text{ for all } s' \preceq t' \in \mathcal{L}(T)\}$$

Below we recursively construct a sequence $\{B^n\}_n$ with each $B^n \in \mathrm{Tab}'[s \preceq t]$.

Initially B^0 is the tableaux tree which contains only a fixed root labeled with $s \preceq t$. Then suppose $B^n \in \mathrm{Tab}'[s \preceq t]$ is constructed. If all leaves $s' \preceq t'$ of B^n are terminal (and successful since $s' \preceq_\kappa t'$), then the construction is ended. Otherwise, we fix arbitrarily a non-terminal leaf v of B^n and the construction of B^{n+1} is divided into two cases below:

1. $\mathcal{L}(v) = pX\alpha \preceq f$ with $\alpha \in \mathcal{E} \cup \{\epsilon\}$. Then $B^{n+1} = \mathcal{B}_v^n[\mathcal{L}(v), \mathcal{G}]$ with
$$\mathcal{G} = \{(q\beta, g) \in \mathrm{Der}(pX\alpha) \times \mathrm{Der}(f) \mid q\beta \preceq_\kappa g\}.$$
2. $\mathcal{L}(v) = pX\alpha \preceq f$ with $\alpha \in \Gamma^+ \cdot (\mathcal{E} + \epsilon)$. Then $B^{n+1} = B_v^n[\mathcal{L}(v), \mathcal{G}]$ with
$$\mathcal{G} = \{pXU \preceq f\} \cup \mathcal{G}_{\alpha,U}^{\mathrm{a}}$$
where U is defined by: $U(q) = \{g \in F \mid q\alpha \preceq_\kappa g\}$ for all $q \in Q$. Following Lemma 29, we obtain $pXU \preceq_\kappa f$.

Then in either case $B^{n+1} \in \mathrm{Tab}'[s \preceq t]$. The construction of $\{B^n\}_n$ ends in finitely many steps. This is shown by contraposition. Suppose this is not the case. Denote $B^n = (T_n, \mathcal{L}_n)$ and $B = (\bigcup_n T_n, \bigcup_n \mathcal{L}_n)$. Then B is an infinite finitely-branching proof tree. Thus by König's Lemma there is an infinite path in B. However, by Lemma 24 on such infinite path there must be v, v' with $v \neq v'$ such that $\mathcal{L}(v) = \mathcal{L}(v')$. By Definition 20, either v or v' is successful, thus the construction should end at v or v'. Contradiction. Then the tableaux tree B^l which is the last element of $\{B^n\}_n$ is a successful tableaux tree. ◀

Below we illustrate our main result, where we use a refinement technique to achieve the EXPTIME-upperbound. We define $|P|$ (resp. $|\mathcal{F}|$) to be the integrated size of Q, Γ, L, Δ (resp. F, A, Ω).

▶ **Theorem 32.** *The problem whether $s \preceq_\kappa t$ for a given $(s,t) \in \Pi[Q \times \Gamma, F]$ is decidable in $\mathcal{O}(H(|P|, |\mathcal{F}|) \cdot 8^{|F||Q|})$ time where H is a fixed multivariate polynomial. Thus, if $|Q|$ and $|F|$ are fixed, then the problem can be decided in PTIME.*

Proof. We assume that $s \preceq t \in (Q \times \Gamma) \times F$ and $\kappa = \mathrm{c}$, the other cases are similar. We present a refinement algorithm to decide if $s \preceq_\mathrm{c} t$. Formally, we construct a finite decreasing sequence of sets of goals $\{\mathcal{G}_n\}_n$ where the last element $\mathcal{G}_m$ is expected to contain all the correct goals in $\mathcal{G}^\mathrm{a}_*$. The construction is as follows: Initially $\mathcal{G}_0 = \mathcal{G}^\mathrm{a}_*$. Then $\mathcal{G}_{n+1} \subseteq \mathcal{G}^\mathrm{a}_*$ is constructed from $\mathcal{G}_n$ as follows: $s \preceq t \in \mathcal{G}_{n+1}$ iff either $s \preceq t$ is basically successful, or $s \preceq t \in \mathcal{G}_n$ and there is $(s \preceq t, \mathcal{G}) \in \mathrm{RULE}_\mathrm{c}$ such that $\mathcal{G} \subseteq \mathcal{G}_n$. Here note that $|\mathcal{G}^\mathrm{a}_*| = \mathcal{O}(|P|^3 |F| 2^{|F||Q|})$.

The computation from $\mathcal{G}_n$ to $\mathcal{G}_{n+1}$ can be done in $\mathcal{O}(H'(|P|, |\mathcal{F}|) \cdot 4^{|F||Q|})$ time where H' is a fixed multivariate polynomial. Given $s \preceq t \in \mathcal{G}_n$, we can check whether $s \preceq t \in \mathcal{G}_{n+1}$ as follows: If $s \preceq t = pX\alpha \preceq f$ with $\alpha \in \Gamma^+ \cdot (\mathcal{E} + \epsilon)$, we check if $\{pXU \preceq f\} \cup \mathcal{G}^\mathrm{a}_{\alpha,U} \subseteq \mathcal{G}_n$ for some $U \in \mathcal{E}$. If $s \preceq t = pX\alpha \preceq f$ with $\alpha \in \mathcal{E} \cup \{\epsilon\}$, we check if for any $pX\alpha \xrightarrow{a}_\mathrm{n} \mu$, there is $f \xrightarrow{a}_\mathrm{c} \nu$ such that $\mu \mathcal{G}_n \nu$; this can be checked by checking if the following linear inequality system (with variables $\{x_\nu\}_{\nu \in \mathrm{der}(f,a)}$ and $\{y_{(s',t')}\}_{(s',t') \in \lfloor \mu \rfloor \times F}$) has a solution:

- $\sum_{\nu \in \mathrm{der}(f,a)} x_\nu = 1$, and $x_\nu \geq 0$ for all $\nu \in \mathrm{der}(f,a)$.
- $\sum_{t' \in F} y_{(s',t')} = \mu(s')$ for all $s' \in \lfloor \mu \rfloor$.
- $\sum_{s' \in \lfloor \mu \rfloor} y_{(s',t')} = \sum_{\nu \in \mathrm{der}(f,a)} x_\nu \cdot \nu(t')$ for all $t' \in F$.
- $y_{(s',t')} \geq 0$ for all $(s',t') \in \lfloor \mu \rfloor \times F$, and $y_{(s',t')} = 0$ for all $(s',t') \notin \mathcal{G}_n$.

This can be solved in polynomial time in $|P|$ and $|\mathcal{F}|$ [17].

Since $\mathcal{G}_{n+1} \subseteq \mathcal{G}_n$ there is $m \leq |\mathcal{G}^\mathrm{a}_*|$ such that $\mathcal{G}_{m+1} = \mathcal{G}_m$. We show that for any $s \preceq t \in \mathcal{G}^\mathrm{a}_*$, $s \preceq_\mathrm{c} t$ iff $s \preceq t \in \mathcal{G}_m$. Suppose $s \preceq_\mathrm{c} t$. Let $(T, \mathcal{L}) \in \mathrm{Tab}'[s \preceq t]$ be the tableaux tree constructed in the proof of Proposition 31. It follows by induction on n that $\mathcal{L}(T) \subseteq \mathcal{G}_n$ for all $n \in \mathbb{N}_0$. Thus $s \preceq t \in \mathcal{G}_m$. Suppose now that $s \preceq t \in \mathcal{G}_m$. Since for any $s' \preceq t' \in \mathcal{G}_m$ which is not basically successful, there is $(s' \preceq t', \mathcal{G}) \in \mathrm{RULE}_\mathrm{c}$ such that $\mathcal{G} \subseteq \mathcal{G}_m$. Thus we can iteratively apply rules to the root $s \preceq t$ and form a successful tableaux tree similar to the construction in the proof of Proposition 31. Thus, $s \preceq_\mathrm{c} t$ by Proposition 28. ◀

▶ Remark. The major difference between our tableaux proof system and Colin Stirling's [19, 20] is at the RED (Reduction) rules: here we need to tackle extended stack symbols in our setting, which is different from Stirling's version. Another difference is that we are able to derive a primitive upperbound by a refinement technique, which is not feasible in [19, 20].

5 EXPTIME-Hardness

In this section we show that deciding $\sqsubseteq_\kappa$ is EXPTIME-hard, whenever $\kappa = \mathrm{n}$ or $\kappa = \mathrm{c}$. We prove this by providing a rather straightforward reduction from the non-probabilistic EXTPIME-hardness result obtained in [14]. Our main efforts lie in the treatment of the additional "$\mathrm{Act}(s) = \mathrm{Act}(t)$" condition in Definition 7 which is not involved in the definition of non-probabilistic simulation preorder. First we define a variation of $\sqsubseteq_\kappa$.

▶ **Definition 33.** Let $\mathcal{T} = (S, A, \Omega)$ be a pTS. Define $\preccurlyeq_\kappa$ to be the union of all binary relations $\mathcal{R} \subseteq S \times S$ such that for any $(s,t) \in \mathcal{R}$, whenever $s \xrightarrow{a}_\mathrm{n} \mu$ there is $t \xrightarrow{a}_\kappa \nu$ with $\mu \mathcal{R} \nu$.

In other words, $\preccurlyeq_\kappa$ is defined in a similar way of $\preceq_\kappa$, however without the "$\mathrm{Act}(s) = \mathrm{Act}(t)$" requirement. Then we embed non-probabilistic transition systems into pTS's.

▶ **Definition 34.** A distribution μ is *Dirac* if $||\lfloor\mu\rfloor|| = 1$. The Dirac distribution μ with $\lfloor\mu\rfloor = \{s\}$ is also written as $\delta[s]$. A pTS (S, A, Ω) is *Dirac* if μ is dirac for any $(s, a, \mu) \in \Omega$. A pPDA (Q, Γ, L, Δ) is *Dirac* if μ is Dirac for any $pX \xrightarrow{a} \mu \in \Delta$.

Note that a Dirac pPDA induces a Dirac pTS. Dirac pTS's correspond to labeled transition systems without probability. It is easy to see that $\preccurlyeq_\mathrm{n}$ is the non-probabilistic simulation preorder over labeled transition systems. By [14], deciding $\preccurlyeq_\mathrm{n}$ is EXPTIME-hard between Dirac pPDA and finite Dirac pTS in both direction. Below we reduce $\preccurlyeq_\kappa$ to $\sqsubseteq_\kappa$ under Dirac pTS's. The following proposition allows us to focus solely on the case $\kappa = \mathrm{n}$.

▶ **Proposition 35.** *If the underlying pTS $\mathcal{T} = (S, A, \Omega)$ is Dirac, then $\preccurlyeq_\mathrm{n}$=$\preccurlyeq_\mathrm{c}$ and $\sqsubseteq_\mathrm{n}$=$\sqsubseteq_\mathrm{c}$.*

Now we reduce $\preccurlyeq_\mathrm{n}$ between a Dirac pPDA $P = (Q, \Gamma, L, \Delta)$ and a Dirac finite pTS $\mathcal{F} = (F, A, \Omega)$, to $\sqsubseteq_\mathrm{n}$ between a Dirac pPDA $(Q', \Gamma', L, \Delta')$ and a Dirac finite pTS (F', A, Ω'). The reduction is as follows:

1. $Q' = Q \cup \{p_\perp\}$ and $F' = F \cup \{f_\perp\}$ where $p_\perp \notin Q$ and $f_\perp \notin F$.
2. $\Gamma' = \Gamma \cup \{Z_\perp\}$ where $Z_\perp \notin \Gamma$ is a new bottom stack symbol.
3. $\Delta' = \Delta \cup \{(pX, a, \delta[p_\perp]) \mid p \in Q, X \in \Gamma', a \in L \cup A\}$
4. $\Omega' = \Omega \cup \{(f, a, \delta[f_\perp]) \mid f \in F, a \in L \cup A\}$.

It is not hard to prove that for all $p\alpha \in Q \times \Gamma^*$ and $f \in F$, $p\alpha \preccurlyeq_\mathrm{n} f$ (resp. $f \preccurlyeq_\mathrm{n} p\alpha$) iff $p\alpha Z_\perp \sqsubseteq_\mathrm{n} f$ (resp. $f \sqsubseteq_\mathrm{n} p\alpha Z_\perp$). Thus deciding $\sqsubseteq_\mathrm{n}$ between Dirac pPDA's and Dirac finite pTS's is EXPTIME-hard. Then:

▶ **Theorem 36.** *Deciding $\sqsubseteq_\mathrm{n}$ and $\sqsubseteq_\mathrm{c}$ between probabilistic pushdown automata and finite probabilistic transition systems in both directions is EXPTIME-complete.*

6 Conclusion

We have shown that deciding probabilistic simulation preorder between a probabilistic pushdown automata (Q, Γ, L, Δ) and a finite probabilistic transition system (F, A, Ω) is EXPTIME-complete. This result holds for both directions. Further if $|Q|$ and $|F|$ are fixed, then the problem is decidable in polynomial time. These results extend their non-probabilistic counterparts in [14]. We obtain these results by extending Colin Stirling's method [19, 20] which is originally used to demonstrate the decidability of bisimulation over pushdown automata. Our extension is nontrivial and has a different form from the original one. A future direction is to explore if this method can be extended to weak semantical equivalences such as weak probabilistic bisimulation or weak probabilistic simulation [3, 18].

Acknowledgement

Thanks to anonymous referees for valuable comments. The first author is supported by a CSC scholarship. The second author is supported by the EU FP7 project MoVeS.

References

1 Christel Baier, Bettina Engelen, and Mila E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.*, 60(1):187–231, 2000.

2 Christel Baier, Holger Hermanns, and Joost-Pieter Katoen. Probabilistic weak simulation is decidable in polynomial time. *Inf. Process. Lett.*, 89(3):123–130, 2004.

3 Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Verena Wolf. Comparative branching-time semantics for Markov chains. *Inf. Comput.*, 200(2):149–214, 2005.

4 Tomás Brázdil, Antonín Kučera, and Oldrich Strazovský. On the decidability of temporal properties of probabilistic pushdown automata. In *STACS*, pages 145–157, 2005.

5 Tomás Brázdil, Antonín Kučera, and Oldrich Strazovský. Deciding probabilistic bisimilarity over infinite-state probabilistic systems. *Acta Inf.*, 45(2):131–154, 2008.

6 Stefano Cattani and Roberto Segala. Decision algorithms for probabilistic bisimulation. In *CONCUR*, pages 371–385, 2002.

7 Javier Esparza, Antonín Kučera, and Richard Mayr. Model checking probabilistic pushdown automata. In *LICS*, pages 12–21, 2004.

8 Kousha Etessami and Mihalis Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *TACAS*, pages 253–270, 2005.

9 Kousha Etessami and Mihalis Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1), 2009.

10 Jan Friso Groote and Hans Hüttel. Undecidable equivalences for basic process algebra. *Inf. Comput.*, 115(2):354–371, 1994.

11 Bengt Jonsson, Kim G. Larsen, and Wang Yi. Probabilistic extensions of process algebras. In *Handbook of Process Algebra*, pages 685–710. Elsevier, 2001.

12 Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277, 1991.

13 Antonín Kučera, Javier Esparza, and Richard Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006.

14 Antonín Kučera and Richard Mayr. On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Inf. Comput.*, 208(7):772–796, 2010.

15 Marta Z. Kwiatkowska. Model checking for probability and time: from theory to practice. In *LICS*, pages 351–360, 2003.

16 Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

17 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

18 Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, pages 481–496, 1994.

19 Colin Stirling. Decidability of bisimulation equivalence for normed pushdown processes. *Theor. Comput. Sci.*, 195(2):113–131, 1998.

20 Colin Stirling. Decidability of bisimulation equivalence for pushdown processes. *Unpublished manuscript, available at http://homepages.inf.ed.ac.uk/cps/*, 2000.

Parameterised Pushdown Systems with Non-Atomic Writes

Matthew Hague

Oxford University, Department of Computer Science, and
Laboratoire d'Informatique Gaspard-Monge, Université Paris-Est

Abstract

We consider the master/slave parameterised reachability problem for networks of pushdown systems, where communication is via a global store using only non-atomic reads and writes. We show that the control-state reachability problem is decidable. As part of the result, we provide a constructive extension of a theorem by Ehrenfeucht and Rozenberg to produce an NFA equivalent to certain kinds of CFG. Finally, we show that the non-parameterised version is undecidable.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Verification, Concurrency, Pushdown Systems, Reachability, Parameterised Systems, Non-atomicity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.457

1 Introduction

A parameterised reachability problem is one where the system is defined in terms of a given input, usually a number n. We then ask whether there is some n such that the resulting system can reach a given state. An early result shows that this problem is undecidable, even when the system defined for each n is a finite state machine: one simply has to define the nth system to simulate a Turing machine up to n steps [2]. Thus, the Turing machine terminates iff there is some n such that the nth system reaches a halting state.

Such a result, however, is somewhat pathological. More natural parameterised problems concentrate on the replication of components. For instance, we may have a leadership election algorithm amongst several nodes. For this algorithm we would want to know, for example, whether there is some n such that, when n nodes are present, the routine fails to elect a leader. This problem walks the line between decidability and undecidability, even with finite-state components: in a ring network, when nodes can communicate to their left and right neighbours directly, Suzuki proves undecidability [32]; but, in less disciplined topologies, the problem becomes decidable [16].

In particular, the above decidability result considers the following problem: given a master process $\mathcal{U}$ and slave $\mathcal{C}$, can the master in parallel with n slaves reach a given state. Communication in this system is by anonymous pairwise synchronisation (that is, a receive request can be satisfied by *any* thread providing the matching send, rather than a uniquely identified neighbour). This problem reduces to Petri-nets, which can, for each state of $\mathcal{C}$, keep a count of the number of threads in that state. When communication is via a finite-state global store, which all threads can read from and write to (atomically), it is easy to see that decidability can be obtained by the same techniques.

These results concern finite-state machines. This is ideal for hardware or simple protocols. When the components are more sophisticated (such as threads created by a web-server), a more natural and expressive (infinite-state) program model — allowing one to

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 457–468

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

accurately simulate the control flow of first-order recursive programs [20] — is given by pushdown systems (PDSs). Such systems have proved popular in the sequential setting (e.g. [8, 14, 29, 27]), with several successful implementations [6, 7, 29]. Unfortunately, when two PDSs can communicate, reachability quickly becomes undecidable [26].

In recent years, many researchers have tackled this problem, proposing many different approximations, and restrictions on topology and communication behaviour (e.g. [23, 9, 10, 11, 30, 28, 18]). A pleasantly surprising (and simple) result in this direction was provided by Kahlon [21]: the parameterised reachability problem for systems composed of n slaves $\mathcal{C}$ communicating by anonymous synchronisation is decidable. This result relies heavily on the inability of the system to restrict the number of active processes, or who they communicate with. Indeed, in the presence of a master process $\mathcal{U}$, or communication via a global store, undecidability is easily obtained.

In this work we study the problem of adding the master process and global store. To regain decidability, we only allow *non-atomic* accesses to the shared memory. We then show — by extending a little-cited theorem of Ehrenfeucht and Rozenberg [13] — that we can replace the occurrences of $\mathcal{C}$ with regular automata[1]. This requires the introduction of different techniques than those classically used. Finally, a product construction gives us our result. In addition, we show that, when n is fixed, the problem remains undecidable, for all n. For clarity, we present the single-variable case here. In the appendix we show that the techniques extend easily to the case of k shared variables.

After discussing further related work, we begin in Section 2 with the preliminaries. In Section 3 we define the systems that we study. Our main result is given is Section 4 and the accompanying undecidability proof appears in Section 5. In Section 6 we show how to obtain a constructive version of Ehrenfeucht and Rozenberg's theorem. Finally, we conclude in Section 7. A version of this paper complete with appendix is available [17].

Related Work Many techniques attack parameterisation (e.g. network invariants and symmetry). Due to limited space, we only discuss PDSs here. In addition to results on parameterised PDSs, Kahlon shows decidability of concurrent PDSs communicating via nested-locks [22]. In contrast, we cannot use locks to guarantee atomicity here.

A closely related model was studied by Bouajjani *et al.* in 2005. As we do, they allow PDSs to communicate via a global store. They do not consider parameterised problems directly, but they do allow the dynamic creation of threads. By dynamically creating an arbitrary number of threads at the start of the execution, the parameterised problem can be simulated. Similarly, parameterisation can simulate thread creation by activating hitherto dormant threads. However, since Bouajjani *et al.* allow atomic read/write actions to occur, the problem they consider is undecidable; hence, they consider *context-bounded reachability*.

Context-bounded reachability is a popular technique based on the observation that many bugs can be identified within a small number of context switches [25]. This idea has been extended to *phase-bounded* systems where only one stack may be decreasing in any one phase [3, 31]. Finally, in another extension of context-bounded model-checking, Ganty *et al.* consider *bounded under-approximations* where runs are restricted by intersecting with a word of the form $a_1^* \dots a_n^*$ [15]. In contrast to this work, these techniques are only accurate up to a given bound. That is, they are sound, but not complete. Recently, La Torre *et al.* gave a sound algorithm for parameterised PDSs together with a technique that may detect

[1] A reviewer points out that the upward-closure of a context free language has been proved regular by Atig *et al.* [5] with the same complexity, which is sufficient for our purposes. However, a constructive version of Ehrenfeucht and Rozenberg is a stronger result, and hence remains a contribution.

completeness in the absence of recursion [34].

Several models have been defined for which model-checking can be sound and complete. For example, Bouajjani *et al.* also consider acyclic topologies [5, 11]. As well as restricting the network structure, Sen and Viswanathan [30], La Torre *et al.* [33] and later Heußner *et al.* [18], show how to obtain decidability by only allowing communications to occur when the stack satisfies certain conditions.

One of the key properties that allow parameterized problems to become decidable is that once a copy of the duplicated process has reached a given state, then any number of additional copies may also be in that state. In effect, this means that any previously seen state may be returned to at any time. This property has also been used by Delzanno *et al.* to analyse recursive ping-pong protocols [12] using *Monotonic Set-extended Prefix Rewriting*. However, unlike our setting, these systems do not have a master process.

Finally, recent work by Abdulla *et al.* considers parameterised problems with non-atomic global conditions [1]. That is, global transitions may occur when the process satisfy a global condition that is not evaluated atomically. However, the processes they consider are finite-state in general. Although a procedure is proposed when unbounded integers are allowed, this is not guaranteed to terminate.

2 Preliminaries

We recall the definitions of finite automata and pushdown systems and their language counter-parts. We also state a required result by Ehrenfeucht and Rozenberg.

▶ **Definition 1** (Non-Deterministic Finite Word Automata)**.** We define a *non-deterministic finite word automaton* (NFA) $\mathcal{A}$ as a tuple $(\mathcal{Q}, \Gamma, \Delta, q_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of states, Γ is a finite alphabet, $q_0 \in \mathcal{Q}$ is an initial state, $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final states, and $\Delta \subseteq \mathcal{Q} \times \Gamma \times \mathcal{Q}$ is a finite set of transitions.

We will denote a transition (q, γ, q') using the notation $q \xrightarrow{\gamma} q'$. We call a sequence $q_1 \xrightarrow{\gamma_1} q_2 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_{z-1}} q_z$ a *run* of $\mathcal{A}$. It is an accepting run if $q_1 = q_0$ and $q_z \in \mathcal{F}$. The language $\mathcal{L}(\mathcal{A})$ of an NFA is the set of all words labelling an accepting run. Such a language is *regular*.

▶ **Definition 2** (Pushdown Systems)**.** A *pushdown system* (PDS) $\mathcal{P}$ is defined as a tuple $(\mathcal{Q}, \Sigma, \Gamma, \Delta, q_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of control states, Σ is a finite stack alphabet with a special bottom-of-stack symbol $\perp$, Γ is a finite output alphabet, $q_0 \in \mathcal{Q}$ is an initial state, $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final states, and $\Delta \subseteq (\mathcal{Q} \times \Sigma) \times \Gamma \times (\mathcal{Q} \times \Sigma^*)$ is a finite set of transition rules.

We will denote a transition rule $((q, a), \gamma, (q', w'))$ using the notation $(q, a) \overset{\gamma}{\hookrightarrow} (q', w')$. The bottom-of-stack symbol is neither pushed nor popped. That is, for each rule $(q, a) \overset{\gamma}{\hookrightarrow} (q', w') \in \Delta$ we have, when $a \neq \perp$, w does not contain $\perp$, and, $a = \perp$ iff $w' = w \perp$ and w does not contain $\perp$. A configuration of $\mathcal{P}$ is a tuple (q, w), where $q \in \mathcal{Q}$ is the current control state and $w \in \Sigma^*$ is the current stack contents. There exists a transition $(q, aw) \xrightarrow{\gamma} (q', w'w)$ of $\mathcal{P}$ whenever $(q, a) \overset{\gamma}{\hookrightarrow} (q', w') \in \Delta$. We call a sequence $c_0 \xrightarrow{\gamma_1} c_1 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_z} c_z$ a *run* of $\mathcal{P}$. It is an accepting run if $c_0 = (q_0, \perp)$ and $c_z = (q, w)$ with $q \in \mathcal{F}$. The language $\mathcal{L}(\mathcal{P})$ of a pushdown system is the set of all words labelling an accepting run. Such a language is *context-free*. Note, in some cases, we omit the output alphabet Γ. In this case, the only character is the empty character ε, with which all transitions are labelled. In general, we will omit the empty character ε when it labels a transition.

We use a theorem of Ehrenfeucht and Rozenberg [13]. With respect to a context-free language $\mathcal{L}$, a *strong iterative pair* is a tuple (x, y, z, u, t) of words such that for all $i \geq 0$ we have $xy^izu^it \in \mathcal{L}$, where y and u are non-empty words. A strong iterative pair is *very degenerate* if, for all $i, j \geq 0$ we have that $xy^izu^jt \in \mathcal{L}$.

▶ **Theorem 3** ([13]). *For a given context-free language $\mathcal{L}$, if all strong iterative pairs are very degenerate, then $\mathcal{L}$ is regular.*

However, Ehrenfeucht and Rozenberg do not present a constructive algorithm for obtaining a regular automaton accepting the same language as an appropriate context-free language. Hence, we provide such an algorithm in Section 6.

3 Non-Atomic Pushdown Systems

Given an alphabet $\mathcal{G}$, let $r(\mathcal{G}) = \{\ r(g) \mid g \in \mathcal{G}\ \}$ and $w(\mathcal{G}) = \{\ w(g) \mid g \in \mathcal{G}\ \}$. These alphabets represent read and write actions respectively of the value g.

▶ **Definition 4** (Non-atomic Pushdown Systems). Over a finite alphabet $\mathcal{G}$, a *non-atomic pushdown system* (naPDS) is a tuple $\mathcal{P} = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{G})$ where $\mathcal{Q}$ is a finite set of control-states, Σ is a finite stack alphabet with a bottom-of-stack symbol $\bot$, $q_0 \in \mathcal{Q}$ is a designated initial control state and $\Delta \subseteq (\mathcal{Q} \times \Sigma) \times (r(\mathcal{G}) \cup w(\mathcal{G}) \cup \{\ \varepsilon\ \}) \times (\mathcal{Q} \times \Sigma^*)$.

That is, a non-atomic pushdown system is a PDS where the output alphabet is used to signal the interaction with a global store, and there are no final states: we are interested in the behaviour of the system, rather than the language it defines.

▶ **Definition 5** (Networks of naPDSs). A network of n *non-atomic pushdown systems* (NPDS) is a tuple $\mathcal{N} = (\mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{G}, g_0)$ where, for all $1 \leq i \leq n$, $\mathcal{P}_i = \left(\mathcal{Q}_i, \Sigma_i, \Delta_i, q_0^i, \mathcal{G}\right)$ is a $NPDS$ over $\mathcal{G}$ and $g_0 \in \mathcal{G}$ is the initial value of the global store.

A configuration of an NPDS is a tuple $(q_1, w_1, \ldots, q_n, w_n, g)$ where $g \in \mathcal{G}$ and for each i, $q_i \in \mathcal{Q}_i$ and $w_i \in \Sigma_i^*$. There is a transition $(q_1, w_1, \ldots, q_n, w_n, g) \to (q_1', w_1', \ldots, q_n', w_n', g')$ whenever, for some $1 \leq i \leq n$ and all $1 \leq j \leq n$ with $i \neq j$, we have $q_j' = q_j$, $w_j' = w_j$, and

- $(q_i, w_i) \to (q_i', w_i')$ is a transition of $\mathcal{P}_i$ and $g' = g$; or
- $(q_i, w_i) \xrightarrow{r(g)} (q_i', w_i')$ is a transition of $\mathcal{P}_i$ and $g' = g$; or
- $(q_i, w_i) \xrightarrow{w(g')} (q_i', w_i')$ is a transition of $\mathcal{P}_i$.

A path π of $\mathcal{N}$ is a sequence of configurations $c_1 c_2 \ldots c_m$ such that, for all $1 \leq i < m$, $c_i \to c_{i+1}$. A run of $\mathcal{N}$ is a path such that $c_1 = \left(q_0^1, \bot, \ldots, q_0^n, \bot, g_0\right)$.

4 The Parameterised Reachability Problem

We define and prove decidability of the parameterised reachability problem for naPDSs. We finish with a few remarks on the extension to multiple variables, and on complexity issues.

▶ **Definition 6** (Parameterised Reachability). For given naPDSs $\mathcal{U}$ and $\mathcal{C}$ over $\mathcal{G}$, initial store value g_0 and control state q, the parameterised reachability problem asks whether there is some n such that the NPDS $\mathcal{N}_n = \left(\mathcal{U}, \underbrace{\mathcal{C}, \ldots, \mathcal{C}}_{n}, \mathcal{G}, g_0\right)$ has a run to some configuration containing the control state q.

In this section, we aim prove the following theorem.

▶ **Theorem 7.** *The parameterised reachability problem for NPDSs is decidable.*

Without loss of generality, we can assume q is a control-state of $\mathcal{U}$ (a $\mathcal{C}$ process can write its control-state to the global store for $\mathcal{U}$ to read). The idea is to build an automaton which describes for each $g \in \mathcal{G}$ the sequences $g_1 \dots g_m \in \mathcal{G}^*$ that need to be read by some $\mathcal{C}$ process to be able to write g to the global store. We argue using Theorem 3 that such *read languages* are regular (and construct regular automata using Lemma 18). Broadly this is because, between any two characters to be read, any number of characters may appear in the store and then be overwritten before the process reads the required character. We then combine the resulting languages with $\mathcal{U}$ to produce a context-free language that is empty iff the control-state q is reachable.

4.1 Regular Read Languages

For each $g \in \mathcal{G}$ we will define a *read-language* $\mathcal{L}_{w(g)}$ which intuitively defines the language of read actions that $\mathcal{C}$ must perform before being able to write g to the global store. Since $\mathcal{C}$ may have to write other characters to the store before g, we use the symbol # as an abstraction for these writes. The idea is that, for any run of the parameterised system, we can construct another run where each copy of $\mathcal{C}$ is responsible for a single particular write to the global store, and $\mathcal{L}_{w(g)}$ describes what $\mathcal{C}$ must do to be able to write g.

To this end, given a non-atomic pushdown system $\mathcal{P}$ we define for each $g \in \mathcal{G}$ the pushdown system $\mathcal{P}_{w(g)}$ which is $\mathcal{P}$ augmented with a new unique control-state f, and a transition $(q,a) \hookrightarrow (f,a)$ whenever $\mathcal{P}$ has a rule $(q,a) \xhookrightarrow{w(g)} (q',w)$. Furthermore, replace all $(q,a) \xhookrightarrow{w(g')} (q',w)$ rules with $(q,a) \xhookrightarrow{\#} (q',w)$ where $\# \notin \mathcal{G}$. These latter rules signify that the global store contents have been changed, and that a new value must be written before reading can continue. This implicitly assumes that $\mathcal{C}$ does not try to read the last value it has written. This can be justified since, whenever this occurs, because we are dealing with the parameterised version of the problem, we can simply add another copy of $\mathcal{C}$ to produce the required write.

We interpret f as the sole accepting control state of $\mathcal{P}_{w(g)}$ and thus $\mathcal{L}\big(\mathcal{P}_{w(g)}\big)$ is the language of reads (and writes) that must occur for g to be written. We then allow any number of (ignored) read and # events[2] to occur. That is, any word in the read language contains a run of $\mathcal{C}$ with any number of additional actions that do not affect the reachability property interspersed. Let $R = \{\, r(g') \mid g' \in \mathcal{G} \,\} \cup \{\, \# \,\}$, we define the read language $\mathcal{L}_{w(g)} \subseteq R^*$ for $w(g)$ as

$$\mathcal{L}_{w(g)} = \left\{\, R^*\gamma_1 R^* \dots R^*\gamma_z R^* \;\middle|\; \gamma_1 \dots \gamma_z \in \mathcal{L}\big(\mathcal{P}_{w(g)}\big) \,\right\} .$$

Note, in particular, that $\gamma_1 \dots \gamma_z \in R^*$.

▶ **Lemma 8.** *For all $g \in \mathcal{G}$, $\mathcal{L}_{w(g)}$ is regular and an NFA $\mathcal{A}$ accepting $\mathcal{L}_{w(g)}$, of doubly-exponential size, can be constructed in doubly-exponential time.*

Proof. Take any strong iterative pair (x,y,z,t,u) of $\mathcal{L}_{w(g)}$. To satisfy the preconditions of Theorem 3, we observe that $xzu \in \mathcal{L}_{w(g)}$ since we have a strong iterative pair. Then, from the definition of $\mathcal{L}_{w(g)}$ we know $xR^*zR^*u \subseteq \mathcal{L}_{w(g)}$ and hence, for all i,j, $xy^izt^ju \subseteq \mathcal{L}_{w(g)}$ as required. Thus $\mathcal{L}_{w(g)}$ is regular. The construction of $\mathcal{A}$ comes from Lemma 18. ◀

[2] Extra # events will not allow spurious runs, as they only add extra behaviours that may cause the system to become stuck. This is because # is never read by a process.

4.2 Simulating the System

We build a PDS that recognises a non-empty language iff the parameterised reachability problem has a positive solution. The intuition behind the construction of $\mathcal{P}_{sys}$ is that, if a collection of $\mathcal{C}$ processes have been able to use the output of $\mathcal{U}$ to produce a write of some g to the global store, then we may reproduce that group of processes to allow as many writes g to occur as needed. Hence, in the construction below, once $q_i \in \mathcal{F}_i$ has been reached, g_i can be written at any later time. The # character is used to prevent sequences such as $r(g)w(g')r(g)$ occurring in read languages, where no process is able to provide the required write $w(g)$ that must occur after $w(g')$. Note that, if we did not use # in the read languages, such sequences could occur because the $w(g')$ would effectively be ignored.

The construction itself is a product construct between $\mathcal{U}$ and the regular automata accepting the read languages of $\mathcal{C}$. The regular automata read from the global variable, writing # when a # action should occur. Essentially, they mimic the behaviour of an arbitrary number of $\mathcal{C}$ processes in their interaction — via the global store — with $\mathcal{U}$ and each other. The value of the global store is held in the last component of the product.

▶ **Definition 9** ($\mathcal{P}_{sys}$)**.** Given an naPDS $\mathcal{U} = \left(\mathcal{Q}_{\mathcal{U}}, \Sigma, \Delta_{\mathcal{U}}, q_0^{\mathcal{U}}, \mathcal{G}\right)$ with initial store value g_0, a control-state $f \in \mathcal{Q}_{\mathcal{U}}$, and, for each $g \in \mathcal{G}$, a regular automaton

$$\mathcal{A}_{w(g)} = \left(\mathcal{Q}_{w(g)}, R, \Delta_{w(g)}, \mathcal{F}_{w(g)}, q_0^{w(g)}\right),$$

we define the PDS $\mathcal{P}_{sys} = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{F})$ where, if $\mathcal{G} = \{\ g_0, \ldots, g_m\ \}$, then

- $\mathcal{Q} = \mathcal{Q}_{\mathcal{U}} \times \mathcal{Q}_{w(g_0)} \times \cdots \times \mathcal{Q}_{w(g_m)} \times (\mathcal{G} \cup \{\ \#\ \})$,
- $q_0 = \left(q_0^{\mathcal{U}}, q_0^{w(g_0)}, \ldots, q_0^{w(g_m)}, g_0\right)$,
- $\mathcal{F} = \{\ f\ \} \times \mathcal{Q}_{w(g_0)} \times \cdots \times \mathcal{Q}_{w(g_m)} \times (\mathcal{G} \cup \{\ \#\ \})$,

and Δ is the smallest set containing all $(q, a) \hookrightarrow (q', w)$ where $q = (q_{\mathcal{U}}, q_0, \ldots, q_m, g)$ and,

- $q' = (q'_{\mathcal{U}}, q_0, \ldots, q_m, g)$ and $(q_{\mathcal{U}}, a) \hookrightarrow (q'_{\mathcal{U}}, w) \in \Delta_{\mathcal{U}}$, or
- $q' = (q'_{\mathcal{U}}, q_0, \ldots, q_m, g)$ and $(q_{\mathcal{U}}, a) \xrightarrow{r(g)} (q'_{\mathcal{U}}, w) \in \Delta_{\mathcal{U}}$, or
- $q' = (q'_{\mathcal{U}}, q_0, \ldots, q_m, g')$ and $(q_{\mathcal{U}}, a) \xrightarrow{w(g')} (q'_{\mathcal{U}}, w) \in \Delta_{\mathcal{U}}$, or
- $q' = (q_{\mathcal{U}}, q_0, \ldots, q'_i, \ldots, q_m, g)$ and $q_i \xrightarrow{r(g)} q'_i \in \Delta_i$, $q_i \notin \mathcal{F}_i$ and $w = a$, or
- $q' = (q_{\mathcal{U}}, q_0, \ldots, q'_i, \ldots, q_m, \#)$ and $q_i \xrightarrow{\#} q'_i \in \Delta_i$, $q_i \notin \mathcal{F}_i$ and $w = a$, or
- $q' = (q_{\mathcal{U}}, q_0, \ldots, q_m, g_i)$, $q_i \in \mathcal{F}_i$ and $w = a$.

The last transition in the above definition — which corresponds to some copy of $\mathcal{C}$ writing g_i to the global store — can be applied any number of times; each application corresponds to a different copy of $\mathcal{C}$, and, since we are considering the parameterised problem, we can choose as many copies of $\mathcal{C}$ as are required.

▶ **Lemma 10.** *The PDS $\mathcal{P}_{sys}$ has a run to some control-state in $\mathcal{F}$ iff the parameterised reachability problem for $\mathcal{U}$, $\mathcal{C}$, $\mathcal{G}$, g_0 and q has a positive solution.*

The full proof of correctness is given in the appendix. To construct a run reaching q from an accepting run of $\mathcal{P}_{sys}$ we first observe that $\mathcal{U}$ is modelled directly. We then add a copy of $\mathcal{C}$ for every individual write to the global component of $\mathcal{P}_{sys}$. These slaves are able to read from/write to the global component finally enabling them to perform their designated write. This is because (a part of) the changes to the global store is in the read language of the required write.

In the other direction, we build an accepting run of $\mathcal{P}_{sys}$ from a run of the parameterised system reaching q. To this end, we observe again that we can simulate $\mathcal{U}$ directly. To

simulate the slaves, we take, for every character $g \in \mathcal{G}$ written to the store, the copy of $\mathcal{C}$ responsible for its first write. From this we get runs of the $\mathcal{A}_{w(g)}$ that can be interleaved with the simulation of $\mathcal{U}$ and each other to create the required accepting run, where additional writes of each g are possible by virtue of $\mathcal{A}_{w(g)}$ having reached an accepting state (hence we require no further simulation for these writes).

Example Let $\mathcal{U}$ perform the actions $r(1)r(2)w(ok)r(f)$ and $\mathcal{C}$ run either $w(1)r(ok)w(go)$ or $w(2)r(go)w(f)$. Let $\mathcal{L}_1, \ldots, \mathcal{L}_4$ denote the following read languages.

$$\mathcal{L}_{w(1)} = \mathcal{L}_{w(2)} = R^* \quad \mathcal{L}_{w(go)} = R^*\#R^*r(ok)R^* \quad \mathcal{L}_{w(f)} = R^*\#R^*r(go)R^*$$

Take two slaves $\mathcal{C}_1$ and $\mathcal{C}_2$ and the run (the subscript denotes the active process):

$$w(1)_{\mathcal{C}_1} r(1)_{\mathcal{U}} w(2)_{\mathcal{C}_2} r(2)_{\mathcal{U}} w(ok)_{\mathcal{U}} r(ok)_{\mathcal{C}_1} w(go)_{\mathcal{C}_1} r(go)_{\mathcal{C}_2} w(f)_{\mathcal{C}_2} r(f)_{\mathcal{U}} \ .$$

This can be simulated by the following actions on the global component of $\mathcal{P}_{sys}$:

$$w(\#)_{\mathcal{L}_3} w(1)_{\mathcal{L}_1} r(1)_{\mathcal{U}} w(\#)_{\mathcal{L}_4} w(2)_{\mathcal{L}_2} r(2)_{\mathcal{U}} w(ok)_{\mathcal{U}} r(ok)_{\mathcal{L}_3} w(go)_{\mathcal{L}_3} r(go)_{\mathcal{L}_4} w(f)_{\mathcal{L}_4} r(f)_{\mathcal{U}} \ .$$

Note, we have scheduled the $w(\#)$ actions immediately before the write they correspond to.

4.3 Complexity and Multiple Stores

We obtain for each $g \in \mathcal{G}$ an automaton $\mathcal{A}_{w(g)}$ of size $\mathcal{O}\left(2^{2^{f(n)}}\right)$ in $\mathcal{O}\left(2^{2^{f(n)}}\right)$ time for some polynomial f (using Lemma 18) where n is the size of the problem description. The pushdown system $\mathcal{P}_{sys}$, then, has $\mathcal{O}\left(2^{2^{f'(n)}}\right)$ many control states for a polynomial f'. It is well known that reachability/emptiness for PDSs is polynomial in the size of the system (e.g. Bouajjani *et al.* [8]), and hence the entire algorithm takes doubly-exponential time. For the lower bound, one can reduce from SAT to obtain an NP-hardness result (as shown in the appendix). Further work is needed to pinpoint the complexity precisely.

The algorithm presented above only applies to a single shared variable. A more natural model has multiple shared variables. We may allow k variables with the addition of k global components $\mathcal{G}_1, \ldots, \mathcal{G}_k$. The main change required is the use of symbols $\#_1, \ldots, \#_k$ rather than simply $\#$ and to build $\mathcal{P}_{sys}$ to be sensitive to which store is being written to (or erased with some $\#_i$). This does not increase the complexity since $n = |\mathcal{G}_1| + \cdots + |\mathcal{G}_k|$ in the above analysis and the cost of the k-product of variables does not exceed the cost of the product of the $\mathcal{A}_{w(g)}$. We give the full details in the appendix. Note that, using the global stores, we can easily encode a PSPACE Turing machine using $\mathcal{U}$, without stack, and an empty $\mathcal{C}$. Hence the problem for multiple variables is at least PSPACE-hard.

5 Non-parameterized Reachability

We consider the reachability problem when the number of processes n is fixed. In the case when $1 \leq n \leq 2$, undecidability is clear: even with non-atomic read/writes, the two processes can organise themselves to overcome non-atomicity. When $n > 2$, it becomes harder to co-ordinate the copies of $\mathcal{C}$. A simple trick recovers undecidability. More formally, then:

▶ **Definition 11** (Non-parameterized Reachability)**.** For given n and $naPDSs$ $\mathcal{U}$ and $\mathcal{C}$ over $\mathcal{G}$, initial store value g_0 and control state q, the non-parameterised reachability problem asks

whether the NPDS $\mathcal{N}_n = \left(\mathcal{U}, \underbrace{\mathcal{C}, \ldots, \mathcal{C}}_{n}, \mathcal{G}, g_0\right)$ has a run to some configuration containing the control state q.

▶ **Theorem 12.** *The non-parameterized reachability problem is undecidable when $n \geq 1$. When $n > 1$, the result holds even when $\mathcal{U}$ is null.*

Proof. We reduce from the undecidability of the emptiness of the intersection of two context-free languages. First fix some $n \geq 2$ and two pushdown systems $\mathcal{P}_1$, $\mathcal{P}_2$ accepting the two languages $\mathcal{L}_1$ and $\mathcal{L}_2$.

We define $\mathcal{C}$ to be the disjunction of $\mathcal{C}_1, \ldots, \mathcal{C}_n$. That is, $\mathcal{C}$ makes a non-deterministic choice of which $\mathcal{C}_i$ to run ($1 \leq i \leq n$). Let $1, \ldots, n, f, !$ be characters not in the alphabet of $\mathcal{L}_1$ and $\mathcal{L}_2$. The process $\mathcal{C}_1$ will execute, for each $\gamma_1 \ldots \gamma_z \in \mathcal{L}_1$, a sequence

$$w(1)r(n)w(\gamma_1)r(!)w(\gamma_2)r(!)\ldots w(\gamma_z)r(!)w(f)\ .$$

It is straightforward to build $\mathcal{C}_1$ from $\mathcal{P}_1$. Similarly, the process $\mathcal{C}_2$ will execute, for each $a_1 \ldots a_m \in \mathcal{L}_2$, a sequence

$$r(1)w(2)r(\gamma_1)w(!)r(\gamma_2)w(!)\ldots r(\gamma_z)w(!)r(f)$$

and move to a fresh control-state q_f. It is straightforward to build $\mathcal{C}_2$ from $\mathcal{P}_2$. The remaining processes for $3 \leq i \leq n$ simply perform the sequence $r(i-1)w(i)$.

The control-state q_f can be reached iff the intersection of $\mathcal{L}_1$ and $\mathcal{L}_2$ is non-empty. To see this, first consider a word witnessing the non-emptiness of the intersection. There is immediately a run of $\mathcal{N}_n$ reaching q_f where each ith $\mathcal{C}$ process behaves as $\mathcal{C}_i$.

In the other direction, take a run of $\mathcal{N}_n$ reaching q_f. First, observe that for each $1 \leq i \leq n$ there must be some copy of $\mathcal{C}$ running $\mathcal{C}_i$. This is because, otherwise, there is some i not written to the global store, and hence all $i' \geq i$, including n, are not written. Then $\mathcal{C}_1$ can never write f and $\mathcal{C}_2$ can never move to q_f. Finally, take the sequence $a_1 \ldots a_m$ written by $\mathcal{C}_1$ (and read by $\mathcal{C}_2$). This word witnesses non-emptiness as required.

In the case when $n = 1$, we simply have $\mathcal{U}$ run $\mathcal{C}_1$ and $\mathcal{C}$ run $\mathcal{C}_2$. ◀

6 Making Ehrenfeucht and Rozenberg Constructive

We show how to make Theorem 3 constructive. To prove regularity, Ehrenfeucht and Rozenberg assign to each word a set of types $\theta(w)$, and prove that, if $\theta(w) = \theta(w')$, then $w \sim w'$ in the sense of Myhill and Nerode [19]. We first show how to decide $\theta(w) = \theta(w')$, and then show how to build the automaton. For the sake of brevity, we will assume familiarity with context-free grammars (CFGs) and their related concepts [19].

For our purposes, we consider a context-free grammar (in Chomsky normal form) G to be a collection of rules of the form $A \to BC$ or $A \to a$, where A, B and C are *non-terminals* and a is a *terminal* in Γ. There is also a designated *start* non-terminal S. A word w is in $\mathcal{L}(G)$ if there is a *derivation-tree* with root labelled by S such that an internal node labelled by A has left- and right-children labelled by B and C when we have $A \to BC$ in the grammar and a leaf node is labelled by a when it has parent labelled by A (with one child) and $A \to a$ is in the grammar. Furthermore w is the *yield* of the tree; that is, w labels the leaves. Note, all nodes must be labelled according to the scheme just described. One can also consider the derivation of w in terms of *rewrites* from S, where the parent-child relationship in the tree gives the requires rewriting steps.

6.1 Preliminaries

We first recall some relevant definitions from Ehrenfeucht and Rozenberg. We write $\#_a(w)$ to mean the number of occurrences of the character a in the word w.

▶ **Definition 13** (Type of a Word). Let Γ be an alphabet and let $x, w \in \Gamma^*$, We say that w is of type x, or that x is a type of w (denoted $\tau(x, w)$) if

1. for every $a \in \Gamma$, $\#_a(x) \leq 1$, and
2. there exists a homomorphism h such that
   a. for every $a \in \Gamma$, $h(a) \in a \cup a\Gamma^* a$, and
   b. $h(x) = w$.

If x satisfies the above, we also say that x is a type in Γ^*.

Given a CFG G in Chomsky normal form, we assume a derivation tree T of G is a labelled tree where all internal nodes are labelled with the non-terminal represented by the node, and all leaf nodes are labelled by their corresponding characters in Γ. Given a derivation tree T, Ehrenfeucht and Rozenberg define a marked tree $\overline{T}$ with an expanded set of non-terminals and terminals. Simultaneously, we will define the spine of a marked tree. Intuitively, we take a path in the tree and mark it with the productions of G that have been used and the directions taken.

Given an alphabet of terminals and non-terminals Σ and a derivation tree T, let $\overline{\Sigma} = \{ (A, B, C, k) \mid k \in \{ 1, 2 \} \wedge A \to BC \in G \} \cup \{ (A, a) \mid A \to a \in G \}$. This is the marking alphabet of G.

▶ **Definition 14** (Spine of a Derivation Tree). Let T be a derivation tree in G and let $\rho = v_0 \dots v_s$ be a path in T where $s \geq 1$, v_0 is the root of T, v_s is a leaf of T and $\ell(v_0), \dots, \ell(v_s)$ are the labels corresponding to nodes of ρ. Now for each node v_j, $0 \leq j \leq s$, change its label to $\overline{\ell}(v_j)$ as follows:

1. if $A \to BC$ is the production used to rewrite the node j (hence $\ell(v_j) = A$) and v_j has a direct descendant to the left of ρ, then $\ell(v_j)$ is changed to $\overline{\ell}(v_j) = (A, B, C, 1)$,
2. if $A \to BC$ is the production used to rewrite the node j and v_j has a direct descendant to the right of ρ, then $\ell(v_j)$ is changed to $\overline{\ell}(v_j) = (A, B, C, 2)$,
3. if $A \to a$ is the production used to rewrite the node j then $\ell(v_j)$ is changed to $\overline{\ell}(v_j) = (A, a)$,
4. $\overline{\ell}(v_s) = \ell(v_s)$.

The resulting tree is called the *marked ρ-version* of T and denoted by $\overline{T}(\rho)$. The word $\overline{\ell}(v_0) \dots \overline{\ell}(v_s)$ is referred to as the *spine* of $\overline{T}(\rho)$ and denoted by $Spine\big(\overline{T}(\rho)\big)$.

We write $\delta(w, z)$ whenever there exists some u such that the word wu has a derivation tree T in G with a path ρ ending on the last character of w and with $Spine\big(\overline{T}(\rho)\big) = z$. Then, we have $\theta(w) = \{ x \mid \delta(w, z) \wedge \tau(x, z) \}$. Intuitively, this is the *spine-type* of w.

Finally, Ehrenfeucht and Rozenberg show that, whenever all strong iterative pairs of G are very degenerate, then $\theta(w) = \theta(w')$ implies $w \sim w'$. Since there are a finite number of types x, we have regularity by Myhill and Nerode.

6.2 Building the Automaton

We show how to make the above result constructive. The first step is to decide $\theta(w) = \theta(w')$ for given w and w'. To do this, from G and some type x, we build G_x which generates all w such that $\delta(w, z)$ holds for some z of type x. Thus $x \in \theta(w)$ iff $w \in \mathcal{L}(G_x)$.

First note that there is a simple (polynomial) regular automaton $\mathcal{A}_x$ recognising, for $x = a_1 \dots a_s$ the language

$$\left(a_1 \cup a_1 \overline{\Sigma}^* a_1\right) \dots \left(a_s \cup a_s \overline{\Sigma}^* a_s\right)$$

and $z \in \mathcal{L}(\mathcal{A}_x)$ iff z is of type x. The idea is to build this automaton into the productions of G to obtain G_x such that all characters to the left (inclusive) of the path chosen by $\mathcal{A}_x$ are kept, while all those to the right are erased.

▶ **Definition 15** (G_x)**.** For a given word type x and CFG G, the grammar G_x has the following production rules:

- all productions in G,
- $A_q \to B_{q'} C_\varepsilon$ for each $A \to BC \in G$ and $q \xrightarrow{(A,B,C,1)} q'$ in $\mathcal{A}_x$,
- $A_q \to B C_{q'}$ for each $A \to BC \in G$ and $q \xrightarrow{(A,B,C,2)} q'$ in $\mathcal{A}_x$,
- $A_q \to a$ for each $A \to a \in G$ and $q \xrightarrow{(A,a)} q'$ in $\mathcal{A}_x$ where q' is a final state,
- $A_\varepsilon \to B_\varepsilon C_\varepsilon$ for each $A \to BC \in G$,
- $A_\varepsilon \to \varepsilon$ for each $A \to a \in G$.

The initial non-terminal is S_{q_0} where S is the initial non-terminal of G and q_0 is the initial state of $\mathcal{A}_x$.

The correctness of G_x is straightforward and hence relegated to the appendix.

▶ **Lemma 16.** *For all w, we have $w \in \mathcal{L}(G_x)$ iff $x \in \theta(w)$.*

▶ **Lemma 17** (Deciding $\theta(w) = \theta(w')$)**.** *For given w and w', we can decide $\theta(w) = \theta(w')$ in $\mathcal{O}\left(2^{f(n)}\right)$ time for some polynomial f where n is the size of G.*

Proof. For a given alphabet $\overline{\Sigma}$, there are $\sum_{r=1}^{m} r!$ types where $m = \left|\overline{\Sigma}\right|$. Since m is polynomial in n, there are $\mathcal{O}\left(2^{f(n)}\right)$ word types. Hence, we simply check $w \in \mathcal{L}(G_x)$ and $w' \in \mathcal{L}(G_x)$ for each type x. This is polynomial for each x, giving $\mathcal{O}\left(2^{f(n)}\right)$ in total. ◀

From this, we can construct, following Myhill and Nerode, the required automaton, using a kind of fixed point construction beginning with an automaton containing the state q_ε from which the equivalence class associated to the empty word will be accepted.

▶ **Lemma 18.** *For a CFG G such that all strong iterative pairs are very degenerate, we can build an NFA $\mathcal{A}$ of $\mathcal{O}\left(2^{2^{f(n)}}\right)$ size in the same amount of time, where n is the size of G.*

Proof. Let G be a CFG such that all strong iterative pairs are degenerate. We build an NFA $\mathcal{A}$ such that $\mathcal{L}(G) = \mathcal{L}(\mathcal{A})$ by the following worklist algorithm.

1. Let the worklist contain only ε (the empty word) and $\mathcal{A}$ have the initial state q_ε.
2. Take a word w from the worklist.
3. If $w \in \mathcal{L}(G)$, make q_w a final state.
4. For each $a \in \Gamma$
   a. if there is no state $q_{w'}$ such that $\theta(wa) = \theta(w')$, add q_{wa} to $\mathcal{A}$ and add wa to the worklist,
   b. take $q_{w'}$ in $\mathcal{A}$ such that $\theta(wa) = \theta(w')$,
   c. add the transition $q_w \xrightarrow{a} q_{w'}$ to $\mathcal{A}$.
5. If the worklist is not empty, go to point 2, else, return $\mathcal{A}$.

Since this follows the Myhill-Nerode construction, using $\theta(w) = \theta(w')$ as a proxy for $w \sim w'$, we have that the algorithm terminates and is correct. Hence, with the observation that there are $\mathcal{O}\left(2^{2^{f(n)}}\right)$ different values of the sets $\theta(w)$, we have the lemma. ◀

7 Conclusions and Future Work

In this work, we have studied the parameterised master/slave reachability problem for pushdown systems with a global store. This provides an extension of work by Kahlon which did not allow a master process, and communication was via anonymous synchronisation; however, this is obtained at the expense of atomic accesses to global variables. Our algorithm introduces new techniques to pushdown system analysis.

An initial inspiration for this work was the study of weak-memory models, which do not guarantee that — in a multi-threaded environment — memory accesses are *sequentially consistent*. In general, if atomic read/writes are permitted, the verification problem is harder (for example, Atig *et al.* relate the finite-state case to lossy channel machines [4]); hence, we removed atomicity as a natural first step. It is not clear how to extend our algorithm to accommodate weak-memory models and it remains an interesting avenue of future work.

Another concern is the complexity gap between the upper and lower bounds. We conjecture that the upper bound can be improved, although we may require a new approach, since the complexity comes from the construction of regular read languages. A related question is whether we can improve the size of the automata $\mathcal{A}_{w(g)}$. Since a PDS of size n can recognise the language $\{ a^{2^n} \}$, we have a read language requiring an exponential number of a characters; hence, the $\mathcal{A}_{w(g)}$ must be at least exponential. It is worth noting that Meyer and Fischer give a language whose *deterministic* regular automaton is doubly-exponential in the size of the corresponding deterministic PDS [24]. However, in the appendix, we provide an example showing that this language is not very degenerate. If the PDS is not deterministic, Meyer and Fischer prove there is no bound, in general, on the relationship in sizes.

Finally, we may also consider applications to recursive ping-pong protocols in the spirit of Delzanno *et al.* [12].

Acknowledgments Nous remercions Jade Alglave pour plusieurs discussions qui ont amorcées ce travail. This work was funded by EPSRC grant EP/F036361/1. We also thank the anonymous reviewers and Ahmed Bouajjani for their helpful remarks.

References

1 P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Handling parameterized systems with non-atomic global conditions. In *VMCAI*, 2008

2 K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters (IPL)*, 1986.

3 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *Developments in Language Theory*, 2008.

4 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, 2010.

5 M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, 2008.

6 T. Ball and S. K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN*, 2000.

7 T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *POPL*, 2002.

8 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, 1997.

9 A. Bouajjani, J. Esparza, S. Schwoon, and J. Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In *FSTTCS*, 2005.

10 A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *SIGPLAN Not.*, 38(1):62–73, 2003.

11 A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. *CONCUR*, 2005.

12 G. Delzanno, J. Esparza, and J. Srba. Monotonic set-extended prefix rewriting and verification of recursive ping-pong protocols. In *ATVA*, 2006.

13 A. Ehrenfeucht and G. Rozenberg. Strong iterative pairs and the regularity of context-free languages. *ITA*, 19(1):43–56, 1985.

14 J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *TACS*, 2001.

15 P. Ganty, R. Majumdar, and B. Monmege. Bounded underapproximations. In *CAV*, 2010.

16 S. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39:675–735, 1992.

17 M. Hague. Parameterised pushdown systems with non-atomic writes. `arXiv:1109.6264v1 [cs.FL]`, 2011.

18 A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FOSSACS*, 2010.

19 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

20 N. D. Jones and S. S. Muchnick. Even simple programs are hard to analyze. *J. ACM*, 24:338–350, April 1977.

21 V. Kahlon. Parameterization as abstraction: A tractable approach to the dataflow analysis of concurrent programs. In *LICS*, 2008.

22 V. Kahlon, F. Ivancic, and A. Gupta. Reasoning about threads communicating via locks. In *CAV*, 2005.

23 R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1998.

24 A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *FOCS*, 1971.

25 S. Qadeer. The case for context-bounded verification of concurrent programs. In *SPIN*, 2008.

26 G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *TOPLAS*, 2000.

27 T. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.*, 58(1-2):206–263, 2005.

28 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, 2005.

29 S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.

30 K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, 2006.

31 A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In *CAV*, 2010.

32 I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28:213–214, July 1988.

33 S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS*, 2008.

34 S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, 2010.

Higher order indexed monadic systems

Didier Caucal[1] **and Teodor Knapik**[2]

1 **CNRS, LIGM-Université Paris-Est**
`caucal@univ-mlv.fr`
2 **ERIM, Université de la Nouvelle Calédonie**
`knapik@univ-nc.nc`

Abstract

A word rewriting system is called monadic if each of its right hand sides is either a single letter or the empty word. We study the images of higher order indexed languages (defined by Maslov) under inverse derivations of infinite monadic systems. We show that the inverse derivations of deterministic level n indexed languages by confluent regular monadic systems are deterministic level $n+1$ languages, and that the inverse derivations of level n indexed monadic systems preserve level n indexed languages. Both results are established using a fine structural study of classes of infinite automata accepting level n indexed languages. Our work generalizes formerly known results about regular and context-free languages which form the first two levels of the indexed language hierarchy.

1998 ACM Subject Classification F.4

Keywords and phrases Higher-order indexed languages, monadic systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.469

1 Introduction

A word rewriting system is a (possibly infinite) set of pairs of words called rules. The rewriting relation $\longrightarrow$ transforms a word xuy into xvy by applying a rule (u, v), leaving unchanged the left and right contexts x and y. This is denoted by $xuy \longrightarrow xvy$. The iteration (or reflexive and transitive closure under composition) of this relation is called the derivation relation and written $\stackrel{*}{\longrightarrow}$. Word rewriting systems form a Turing-complete model of computation, which implies in particular that the reachability problem 'Given words u and v, is there a derivation from u to v?' is in general undecidable. It becomes however decidable for certain subclasses of *monadic* systems, i.e. systems in which the right hand side of any rule is either a single letter or the empty word [4]. Monadic systems form an important class generalizing the well-known Dyck system, which we used in [10] to provide a decomposition technique for word rewriting systems and generalize existing language preservation properties. The current work finds a direct application in further exploiting this decomposition technique (see the conclusion).

Given a family of languages F, we call a system F-monadic whenever the set of left hand sides of rules with the same right hand side forms a language in F (i.e. the inverse single-step rewriting of any letter or the empty word is a language in F). As can be seen by adapting the saturation method provided in [2], the (image under the) derivation of a regular language by any F monadic system is also regular, and can be effectively computed whenever the emptiness of the intersection of any language in F with a regular language is decidable. This is the case for instance of regular and context-free monadic systems [15, 3], but can be easily generalized to higher-order indexed monadic systems of any level (where levels 0 and 1 correspond to regular and context-free languages; see [13] for a definition of

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 469–480

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

indexed languages). When effective, this regularity preservation property directly implies the decidability of the reachability problem. It is also natural to ask whether this preservation property still holds for classes of indexed languages above level 0 *i.e.* above regular languages, but it turns out this is not the case: the derivation of a context-free language by a finite monadic system can be non-recursive [3].

The situation is quite different when considering the inverse derivation relations of monadic systems. Given a rewriting system R, we denote by $\mathrm{Pre}^*_R(L)$ the set of all words which can be derived by R into a word in L, i.e. the image of L by the inverse derivation of R. In contrast to the above results, when R is a confluent finite monadic system and L is a regular set of R-irreducible words, then $\mathrm{Pre}^*_R(L)$ is a deterministic context-free (and in general non-regular) language [3]. Moreover, when L is a context-free language and R a context-free monadic system, $\mathrm{Pre}^*_R(L)$ is also context-free [3], in other words for context-free monadic systems the operator $\mathrm{Pre}^*_R(L)$ effectively preserves context-freeness. In this work, we generalize these two results to all higher levels of indexed languages.

This work relies on an automata-theoretic characterization of level n indexed languages by automata with n-nested pushdown stores (i.e. 'stacks of stacks'). We call these *level n automata* [14]. We first show that for any confluent regular monadic system R, and any deterministic level n indexed language L, $\mathrm{Pre}^*_R(L)$ is a deterministic level $n+1$ indexed language (Theorem 17). This is done using the notion of Cayley automaton, in which states correspond to R-irreducible words, there is an a-labelled edge from u to v if and only if v is the normal form of ua, and words in the n indexed language L are seen as accepting states. This automaton is a deterministic level $n+1$ automaton recognizing the language $\mathrm{Pre}^*_R(L)$ which is thus a deterministic level $n+1$ indexed language (Proposition 16).

Moreover, we show that for any level n (other than 0), the inverse derivation of any level n indexed monadic system R preserves level n indexed languages (Theorem 22). From any mapping h associating to each right hand side a of R a level n automaton recognizing the set $R^{-1}(a)$ of the left hand sides producing a, we define the *iterated substitution* h^* which transforms any level n automaton recognizing a language L into a level n automaton recognizing $\mathrm{Pre}^*_R(L)$.

This work is organized as follows. In Section 2 we recall the necessary definitions, in particular concerning Thue systems and Cayley graphs. In Section 3, we define a class of graph transformations called inverse regular path functions, a technical tool of independent interest generalizing the notion of inverse regular mapping. Finally in Section 4, we present our main results concerning the inverse derivations of monadic systems.

2 Thue systems and Cayley graphs

We say that a system is canonical if each word derives into a unique irreducible word. To any canonical Thue system R is associated its Cayley graph, which recognizes from ε to any set L of irreducible words, the inverse derivation of L (Proposition 4).

2.1 Graphs

Let T be an infinite countable set of symbols called *terminals*. A *graph* G is a set of edges labelled over T *i.e.* $G \subseteq V \times T \times V$ where V is an arbitrary set such that the following set of *vertices*:

$$V_G = \{ s \in V \mid \exists\, a \in T\ \exists\, t \in V\ (s,a,t) \in G \ \vee\ (t,a,s) \in G \}$$

is finite or countable, and the following set of *labels*:

$$T_G \;=\; \{\; a \mid \exists\; s,t\; (s,a,t) \in G \;\}$$

is finite. A triple $(s,a,t) \in G$ is an *edge* labelled by a from *source* s to *target* t; it is identified with the labelled transition $s \xrightarrow{a}_G t$ or directly $s \xrightarrow{a} t$ if G is understood. Any tuple $(s_0, a_1, s_1, \ldots, a_n, s_n)$ for $n \geq 0$ and $s_0 \xrightarrow{a_1}_G s_1\,, \ldots, s_{n-1} \xrightarrow{a_n}_G s_n$ is a *path* from s_0 to s_n labelled by $u = a_1 \ldots a_n$; we write $s_0 \overset{u}{\Longrightarrow}_G s_n$ or directly $s_0 \overset{u}{\Longrightarrow} s_n$ if G is understood. The *language recognized* by a graph G from a vertex subset $I \subseteq V_G$ to a vertex subset $F \subseteq V_G$ is the label set $\mathrm{L}(G,I,F)$ of all paths from I to F:

$$\mathrm{L}(G,I,F) \;=\; \{\; u \in T_G^* \mid \exists\; i \in I \;\; \exists\; f \in F \;\; (i \overset{u}{\Longrightarrow}_G f) \;\}.$$

A *regular language* is any language recognized by a finite graph; we denote $\mathrm{Reg}(N^*)$ the set of regular languages over $N \subseteq T$. A graph is *deterministic* if it has no two edges with the same source and the same label: $(r \xrightarrow{a} s \;\wedge\; r \xrightarrow{a} t) \implies s = t$. Fixing an alphabet $N \subset T$, a graph G is *N-complete* if $T_G = N$ and for any $a \in N$, every vertex $s \in V_G$ is source of an edge labelled by a : $\exists\; t\; (s \xrightarrow{a} t)$.

2.2 Thue systems

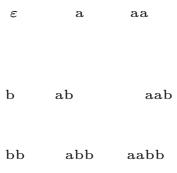
A *Thue system* R over an alphabet $N \subset T$ is a (not necessarily finite) subset of $N^* \times N^*$. Any element $(u,v) \in R$, also denoted by $u\; R\; v$, is a *rule* of R with left hand side (l.h.s. for short) u and right hand side (r.h.s. for short) v. By interverting left and right hand sides of R, we get the *inverse* $R^{-1} = \{\; (v,u) \mid u\; R\; v \;\}$ of R. The *domain* of R is the set $\mathrm{Dom}_R = \{\; u \mid \exists\; v\; (u\; R\; v) \;\}$ and its *range* is the set $\mathrm{Ran}_R = \mathrm{Dom}_{R^{-1}}$. The *identity* relation over a language L is the system $\mathrm{Id}_L = \{\; (u,u) \mid u \in L \;\}$. Given systems R and S, their *concatenation* is $R.S = \{\; (ux, vy) \mid u\; R\; v \;\wedge\; x\; S\; y \;\}$ and their *composition* is $R \circ S = \{\; (u,w) \mid \exists v\; (u\; R\; v \wedge v\; S\; w) \;\}$. The *left concatenation* (resp. *right concatenation*) of a system R by a language $L \subseteq N^*$ is the system $L.R = \mathrm{Id}_L.R = \{\; (xu, xv) \mid x \in L \wedge u\; R\; v \;\}$ (resp. $R.L = R.\mathrm{Id}_L$). A *congruence* R is an equivalence relation on N^* which is closed under left and right concatenation with N^* *i.e.* R is an equivalence such that $R.R \subseteq R$.
The *rewriting* of a system R is the relation $\to_R = N^*.R.N^*$ *i.e.* $xuy \to_R xvy$ for some rule $u\; R\; v$ with left and right contexts $x, y \in N^*$. For any language $L \subseteq N^*$, $\mathrm{Pre}_R(L) = \{\; u \mid \exists\; v \in L\; (u \to_R v) \;\}$ is the set of *predecessors* of L, and $\mathrm{Post}_R(L) = \{\; v \mid \exists\; u \in L\; (u \to_R v) \;\}$ is the set of *successors* of L. The *derivation* $\to_R^*$ by R is the reflexive and transitive closure of $\to_R$ under composition. For any language L, $\mathrm{Pre}_R^*(L) = \{\; u \mid \exists\; v \in L\; (u \to_R^* v) \;\}$ is the set of *ascendants* of L, and $\mathrm{Post}_R^*(L) = \{\; v \mid \exists\; u \in L\; (u \to_R^* v) \;\}$ is the set of *descendants* of L. We denote by $\mathrm{Irr}_R = \{\; u \in N^* \mid \neg\; \exists v\; (u \to_R v) \;\} = N^* - N^* \mathrm{Dom}_R N^*$ the set of *irreducible words* of R. The *Thue congruence* $\leftrightarrow_R^* \;=\; \to_{R \,\cup\, R^{-1}}^*$ is the finest congruence containing R, and we denote by $[u]_{\leftrightarrow_R^*}$ the Thue *congruence class* of $u \in N^*$. The *word problem* for R is, given words u and v, to decide whether $u \leftrightarrow_R^* v$.
We say that a system R is *terminating* if each word derives to an irreducible word: $\forall\; u \in N^*\; \exists\; v \in \mathrm{Irr}_R\; (u \to_R^* v)$. Recall that R is *noetherian* if there is no infinite rewriting chain $u_0 \to_R u_1 \to_R \ldots$ So any noetherian system is terminating but for $a, b \in N$, the system $\{(a,a)\,, (a,b)\}$ is terminating but not noetherian. A system R is *confluent* if every pair of words with a common ancestor have a common descendant: if $\mathrm{Pre}_R^*(u) \cap \mathrm{Pre}_R^*(v) \neq \emptyset$ then $\mathrm{Post}_R^*(u) \cap \mathrm{Post}_R^*(v) \neq \emptyset$. A *canonical system* R is a terminating and confluent system which is equivalent to the condition that each word u derives into a unique irreducible word $u{\downarrow_R}$ called the *normal form* of u. In that case, the congruence class of any word is the set of ascendants of its normal form.

▶ **Lemma 1.** *For any canonical system* R*, we have*

$$\begin{array}{ll} [L]_{\leftrightarrow_R^*} \;=\; \mathrm{Pre}_R^*(L{\downarrow_R}) & \textit{for any } L \subseteq N^*, \\ \{\; [L]_{\leftrightarrow_R^*} \mid L \subseteq N^* \;\} & \textit{is a boolean algebra.} \end{array}$$

2.3 Cayley graphs

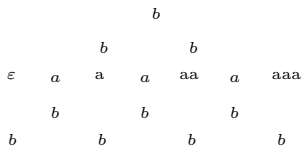
Let us begin with an elementary example. For letters a and b, the finite system $R_0 = \{(a,\varepsilon)\,,\,(b,\varepsilon)\}$ is canonical: ε is the normal form of any word. The rewriting $\to_{R_0}$ restricted to the words in a^*b^* is the following grid:



which has an undecidable monadic second order (MSO) theory, an even an undecidable FO* theory [19] where FO* denotes the first order logic extended with the transitive closure operator of arity one and without parameter. The Thue systems constitute a Turing-complete model of computation, hence their rewritings define a large family of graphs [8] having (by Rice's theorem) strong undecidability results. Instead of considering the rewriting $\to_R$ of any Thue system R, [5] defines the *Cayley graph* of R as

$$[R] \;=\; \{\; u \xrightarrow{a} v \mid u,v \in \mathrm{Irr}_R \;\wedge\; a \in N \;\wedge\; ua \;\to_R^*\; v \;\}.$$

This is inspired by the analogous notion for groups. The Cayley graph of R_0 is $[R_0] = \{\varepsilon \xrightarrow{a} \varepsilon\,,\,\varepsilon \xrightarrow{b} \varepsilon\}$ and the Cayley graph $[R_1]$ of the noetherian system $R_1 = \{(ab,b)\,,\,(b,\varepsilon)\}$ is depicted as follows:



This graph is prefix-recognizable hence it has a decidable MSO theory [7].
Note that $[R] = \emptyset \iff \mathrm{Irr}_R = \emptyset \iff \varepsilon \in \mathrm{Dom}_R$, and that $[R]$ contains the tree

$$\{\; u \xrightarrow{a} ua \mid u \in N^* \;\wedge\; a \in N \;\wedge\; ua \in \mathrm{Irr}_R \;\}$$

hence $V_{[R]} = \mathrm{Irr}_R$. The Cayley graphs of canonical systems are deterministic and complete.

▶ **Lemma 2.** *For any system* R *over* N*,*

$$\begin{array}{rcl} R \textit{ is terminating} & \Longrightarrow & [R] \textit{ is } N\textit{-complete,} \\ R \textit{ is confluent} & \Longrightarrow & [R] \textit{ is deterministic.} \end{array}$$

Let us express the path labels of Cayley graphs of canonical systems.

▶ **Lemma 3.** *For any canonical system* R*,*

$$u \overset{v}{\Longrightarrow}_{[R]} w \iff uv \;\to_R^*\; w \quad \textit{for every } u,w \in \mathrm{Irr}_R \textit{ and } v \in N^*.$$

The set of path labels of the Cayley graph of any canonical system from vertex ε to any vertex subset F is the set of ascendants of words in F.

▶ **Proposition 4.** *For any canonical system* R *and any* $F \subseteq \mathrm{Irr}_R$ *,*

$$\mathrm{L}([R], \varepsilon, F) \;=\; \mathrm{Pre}^*_R(F) \;=\; [F]_{\leftrightarrow^*_R} .$$

Note that the Cayley graph of the empty relation is the N-complete tree:

$$[\emptyset] \;=\; \{\; u \xrightarrow{a} ua \mid u \in N^* \;\wedge\; a \in N \;\}.$$

Let us show how to construct $[R]$ from $[\emptyset]$ for a general system R.
Recall that the *suffix rewriting* $\longrightarrow\!\!\!\!\!\rightarrow_R \;=\; N^*.R$ of any system R is the binary relation on N^* defined by $xu \longrightarrow\!\!\!\!\!\rightarrow_R xv$ *i.e.* the application of a rule $u\, R\, v$ under any left context $x \in N^*$ (the right context being empty). The *suffix derivation* $\longrightarrow\!\!\!\!\!\rightarrow^*_R$ is the reflexive and transitive closure under composition of the suffix rewriting. We say that a system R is *suffix* if

$$\mathrm{Post}_R(\mathrm{Irr}_R.N) \;\subseteq\; \{\varepsilon\} \cup \mathrm{Irr}_R.N\,.$$

Note that this condition is effective for any finite system R and more generally for any *recognizable system*: $R \;=\; U_1{\times}V_1 \cup \ldots \cup U_n{\times}V_n$ for some $n \geq 0$ and $U_1, V_1, \ldots, U_n, V_n \in \mathrm{Reg}(N^*)$. In that case, Dom_R is regular, hence Irr_R and $\mathrm{Post}_R(\mathrm{Irr}_R.N)$ are regular languages. The Cayley graph of a suffix system can be obtained by the suffix derivation.

▶ **Lemma 5.** *For any suffix system* R*,*

$$ua \longrightarrow^*_R v \quad\Longleftrightarrow\quad ua \longrightarrow\!\!\!\!\!\rightarrow^*_R v \quad \textit{for any}\;\; u, v \in \mathrm{Irr}_R \;\textit{and}\; a \in N.$$

In the next section, we introduce a class of graph transformations allowing us to construct from $[\emptyset]$ the Cayley graph $[R]$ of any recognizable suffix system R.

3 Path functions

We introduce a generalization of the notion of inverse regular mapping introduced in [7], called inverse path function. We show that the Cayley graph of any recognizable suffix system can be obtained from the complete and deterministic tree by an inverse path function (Proposition 7).

Let $T_\varepsilon \;=\; T \cup \{\varepsilon\}$ and $F \;=\; \{^{-}, \neg, \vee, \wedge, \cdot, {}^*\}$. We define the set Exp of boolean path *expressions* as the smallest language over $T_\varepsilon \cup F \cup \{(\,,\,)\}$ such that $T_\varepsilon \subseteq \mathrm{Exp}$ and

$$\overline{u}\,,\; (\neg\, u)\,,\; (u \vee v)\,,\; (u \wedge v)\,,\; (u \cdot v)\,,\; (u^*) \in \mathrm{Exp} \quad \text{for any } u, v \in \mathrm{Exp}.$$

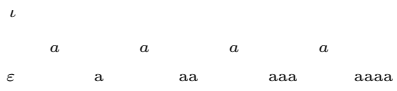
The word label w of a path $s \overset{w}{\Longrightarrow}_G t$ from s to t of a graph G is extended to a regular expression $u \in \mathrm{Exp}$ by induction on the length of u as follows. For any $a \in T$ and $u, v \in \mathrm{Exp}$,

$$\begin{array}{llll}
s \overset{a}{\Longrightarrow} t & \text{if} & s \xrightarrow{a} t & ; \\
s \overset{\overline{u}}{\Longrightarrow} t & \text{if} & t \overset{u}{\Longrightarrow} s & ; \\
s \overset{(u \vee v)}{\Longrightarrow} t & \text{if} & s \overset{u}{\Longrightarrow} t \;\vee\; s \overset{v}{\Longrightarrow} t & ; \\
s \overset{(u \cdot v)}{\Longrightarrow} t & \text{if} & \exists\, r\; (s \overset{u}{\Longrightarrow} r \;\wedge\; r \overset{v}{\Longrightarrow} t) & ;
\end{array}
\qquad
\begin{array}{llll}
s \overset{\varepsilon}{\Longrightarrow} t & \text{if} & s = t & \\
s \overset{(\neg\, u)}{\Longrightarrow} t & \text{if} & \neg\; (s \overset{u}{\Longrightarrow} t) & \\
s \overset{(u \wedge v)}{\Longrightarrow} t & \text{if} & s \overset{u}{\Longrightarrow} t \;\wedge\; s \overset{v}{\Longrightarrow} t & \\
s \overset{(u^*)}{\Longrightarrow} t & \text{if} & s\; (\overset{u}{\Longrightarrow})^*\; t. &
\end{array}$$

For instance $s \overset{(\varepsilon \,\wedge\, (a\,.\,\overline{a}))}{\Longrightarrow} t$ means that $s = t \;\wedge\; \exists\, r\, (s \xrightarrow{a} r)$.
We can remove parentheses using the associativity of $\vee, \wedge, \cdot$ and by assigning priorities to operators as usual. Finally $\cdot$ can be omitted.
A function $h : T \longrightarrow \mathrm{Exp}$ of finite domain is called a regular *path function* and is applied by inverse to any graph G to get the graph:

$$h^{-1}(G) \;=\; \{\; s \xrightarrow{a} t \mid a \in \mathrm{Dom}(h) \;\wedge\; s \overset{h(a)}{\Longrightarrow}_G t \;\}.$$

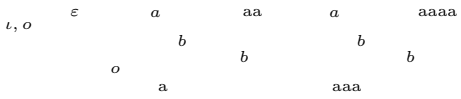
For instance, the path function h defined by $h(a) = a$ and $h(\iota) = a\overline{a} \;\wedge\; \neg(\overline{a}a)$ applied by inverse to the previous Cayley graph $[R_1]$ gives the following graph $h^{-1}([R_1])$:



By applying to this graph the inverse of the path function g defined by

$$\begin{array}{lllllll} g(\iota) & = & \iota & ; & g(a) & = & (\varepsilon \wedge \overline{a}^*\,\iota\,(aa)^*)\,a\,a \\ g(o) & = & \iota \vee \overline{a}\,\iota\,a & ; & g(b) & = & (\varepsilon \wedge (\overline{a}\,\overline{a})^*\,\iota\,a^*)\,\overline{a} \;\vee\; (\varepsilon \wedge (\overline{a}\,\overline{a})^*\overline{a}\,\iota\,a^*)\,\overline{a}\,\overline{a} \end{array}$$

we get the following graph $K = g^{-1}(h^{-1}([R_1]))$ depicted as follows:



where $\mathrm{L}(K, \varepsilon, \{\varepsilon, a\}) \cap \{a, b\}^* \;=\; \{\; a^n b^n \mid n \geq 0 \;\}$.
Inverse path functions are closed under composition. More precisely any path function $h : T \longrightarrow \mathrm{Exp}$ is extended by morphism to a function $\mathrm{Exp} \longrightarrow \mathrm{Exp}$. The expression $h(u)$ is also denoted by $u[h(a_1)/a_1, \ldots, h(a_p)/a_p]$ for $\{a_1, \ldots, a_p\} = \mathrm{Dom}(h)$ and is only defined when $\mathrm{Ele}(u) \cap T \;\subseteq\; \mathrm{Dom}(h)$. This allows to define the *composition* $g \circ h$ of path functions g and h by $(g \circ h)(a) = h(g(a))$ for any $a \in \mathrm{Dom}(g)$ with $g(a) \in \mathrm{Dom}(h)$.
The family of inverse path functions is closed under composition.

▶ **Lemma 6.** *For any graph G and any path functions g and h, we have*

$$g^{-1}(h^{-1}(G)) \;=\; (g \circ h)^{-1}(G).$$

For any recognizable suffix system, we can construct its Cayley graph.

▶ **Proposition 7.** *For any recognizable suffix system R, we can construct a path function h such that $[R] = h^{-1}([\emptyset])$.*

Let us combine Propositions 4 and 7.

▶ **Corollary 8.** *For any recognizable canonical suffix system R and for any regular language $L \subseteq \mathrm{Irr}_R$, $\mathrm{Pre}^*_R(L) \;=\; [L]_{\leftrightarrow^*_R}$ is a deterministic context-free language.*

This follows from the fact that a deterministic prefix-recognizable graph recognizes, from a vertex to a regular vertex set, a deterministic context-free language [7].

4 Monadic systems

We review language preservation properties of the derivation and inverse derivation relations of regular and context-free monadic systems. We generalize these results to higher-order indexed monadic systems using the Shelah-Stupp and Muchnik iterations together with inverse path functions.

4.1 Regular and context-free monadic systems

A system R is *monadic* if ε is not a l.h.s. and any r.h.s. is either a single letter or ε *i.e.* $R \subseteq N^+ \times N_\varepsilon$ for $N_\varepsilon = N \cup \{\varepsilon\}$. Contrary to the usual definition of monadic systems [15, 3, 4], we allow *unitary rules* $a \to b$ for $a, b \in N$. Hence a monadic system R is not in general noetherian. However and in a standard way, we consider the equivalence $\sim$ on N defined for any $a, b \in N$ by $a \sim b$ if $a \to_R^* b \to_R^* a$. We take a mapping $^-$ from N into T such that $\overline{a} = \overline{b} \iff a \sim b$, that we extend by morphism from N^* into T^*. So $\overline{R} = \{ (\overline{u}, \overline{v}) \mid u\, R\, v \wedge \overline{u} \neq \overline{v} \}$ is a monadic system over $\overline{N} = \{ \overline{a} \mid a \in N \}$ such that for any $u, v \in N^*$ ($u \to_R^* v \iff \overline{u} \to_{\overline{R}}^* \overline{v}$). The system $\overline{R}$ can still have unitary rules but $\overline{R}$ is noetherian, and R is confluent $\iff$ $\overline{R}$ is confluent.
We say that a monadic system R is finite (resp. regular, context-free) if for each $a \in N_\varepsilon$, the language $R^{-1}(a)$ of the l.h.s. producing a is finite (resp. regular, context-free). All these subclasses of monadic systems are *effective* in the sense that for each r.h.s. $a \in N_\varepsilon$ we can decide whether $R^{-1}(a) \cap L = \emptyset$ with $L \in \mathrm{Reg}(N^*)$. Note that a monadic system is recognizable if and only if it is regular. A particular finite monadic system is the *Dyck system*: $D = \{ (a\,\overline{a}, \varepsilon) \mid a \in N \} \cup \{ (\overline{a}\,a, \varepsilon) \mid a \in N \}$ where $\overline{a}$ is a new letter for each $a \in N$. The operator Post_D^* preserves regularity: $L \in \mathrm{Reg}(N^*) \implies \mathrm{Post}_D^*(L) \in \mathrm{Reg}(N^*)$. This property has been established in [2] with a saturation method that can be extended to any monadic system.

▶ **Theorem 9.** *For any monadic system R, the operator Post_R^* preserves regularity, and effectively when R is effective.*

This effective regularity preservation has been given for the context-free monadic systems [3] (Theorem 2.5). Let us apply Theorem 9 on $\overline{R}$ when R is confluent.

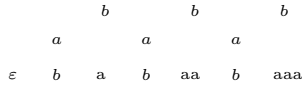
▶ **Corollary 10.** *The word problem is decidable for any effective confluent monadic system.*

The confluence property is decidable for regular monadic systems [15] but is undecidable for context-free monadic systems [3]. Furthermore Post_D^* for the Dyck system D does not preserve context-freeness [12]. In fact $\mathrm{Post}_R^*(L)$ may not be recursive when L is context-free, even if R is a confluent finite monadic system [3] (Theorem 4.1).

We will thus focus on preservation properties of Pre_R^* for monadic systems R. Note that Pre_R^* does not preserve regularity: for the finite monadic system $R = \{(ab, \varepsilon)\}$, we have $\mathrm{Pre}_R^*(\varepsilon) \cap a^*b^* = \{ a^n b^n \mid n \geq 0 \}$ hence $\mathrm{Pre}_R^*(\varepsilon)$ is not regular. However any monadic system is suffix, hence we can apply Corollary 8 on $\overline{R}$ for R confluent.

▶ **Corollary 11.** *For any confluent regular monadic system R and any regular language $L \subseteq \mathrm{Irr}_R$, the set $\mathrm{Pre}_R^*(L)$ is a deterministic context-free language.*

This was already known for the restricted case of finite confluent monadic systems [3] (Theorem 3.9) and of unequivocal monadic systems [15]. Note that the confluence assumption in Corollary 11 cannot be dropped: let $R_2 = \{(ab, \varepsilon), (aab, \varepsilon)\}$ whose Cayley graph restricted to the vertices in a^* is the following non deterministic graph:



The language $\mathrm{Pre}_{R_2}^*(\varepsilon) \cap a^*b^* = \{ a^m b^n \mid n \leq m \leq 2n \}$ is context-free but not deterministic context-free [20], hence $\mathrm{Pre}_{R_2}^*(\varepsilon)$ is not a deterministic context-free language. However $\mathrm{Pre}_{R_2}^*(\varepsilon)$ is context-free. In fact, the inverse of a finite monadic system is a context-free

grammar allowing ε as a l.h.s., and we know that the expressive power of context-free grammars is not increased when allowing a context-free set of r.h.s. for each l.h.s.

▶ **Proposition 12.** [3] *For any context-free monadic system R, the operator Pre^*_R effectively preserves context-freeness.*

We propose to generalize Corollary 11 and Proposition 12 to a hierarchy of monadic systems whose first two levels are the regular and context-free monadic systems.

4.2 Higher-order indexed monadic systems

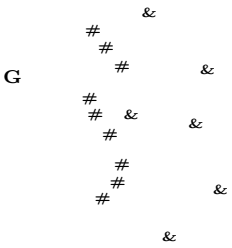
Level n *indexed languages* were introduced for $n = 2$ by Aho et al. [1], and for arbitrary n by Maslov [13]; level 0 and level 1 indexed languages are the regular and context-free languages. These classes of languages coincide with the OI hierarchy of [11]. A monadic system R is n-indexed if for each $a \in N_\varepsilon$, the language $R^{-1}(a)$ is n-indexed; in that case, R is effective [14] and by Theorem 9, Post^*_R effectively preserves regularity.
The n-indexed languages are the languages recognized by automata using an n-nested pushdown store [14] and called *level n automata*. We can describe level $n+1$ automata from level n automata using two basic graph transformations [6]: the previously defined inverse path functions and the full iteration defined by Muchnik [16]. This operation is a generalization of the *basic iteration* $G^\#$ of a graph G with a new label $\# \in T - T_G$ defined by Shelah and Stupp [17, 18]:

$$\begin{aligned} G^\# \;=\;& \{\, (s_1,\ldots,s_n,s) \xrightarrow{a} (s_1,\ldots,s_n,t) \mid n \geq 0 \;\wedge\; s_1,\ldots,s_n \in V_G \;\wedge\; s \xrightarrow{a}_G t \,\} \\ \cup\;& \{\, (s_1,\ldots,s_n) \xrightarrow{\#} (s_1,\ldots,s_n,s) \mid n \geq 0 \;\wedge\; s_1,\ldots,s_n,s \in V_G \,\}. \end{aligned}$$

Muchnik extended this basic iteration to the *full iteration* $G^{\#,\&}$ by marking with a loop labelled by $\& \in T - (T_G \cup \{\#\})$, in each copy of G in $G^\#$, the vertex from which the copy originates:

$$G^{\#,\&} \;=\; G^\# \;\cup\; \{\, (s_1,\ldots,s_n,s,s) \xrightarrow{\&} (s_1,\ldots,s_n,s,s) \mid n \geq 0 \;\wedge\; s_1,\ldots,s_n,s \in V_G \,\}.$$

We give below an illustration of the full iteration of a 'triangle'.



By iteratively applying from the family F_0 of finite graphs the full iteration followed by an inverse path function, we get a hierarchy of graphs [9]: for every $n \geq 0$,

$$\mathrm{F}_{n+1} \;=\; \{\, h^{-1}(G^{\#,\&}) \mid G \in \mathrm{F}_n \;\wedge\; \#,\& \in T - T_G \;\wedge\; h \text{ path function} \,\}.$$

Since inverse path functions are particular MSO-interpretations and the full iteration preserves the decidability of the monadic theory [16, 18], all graphs in this hierarchy have a decidable MSO theory. By Lemma 6, each family F_n is closed under inverse path functions. For $n \neq 0$, F_n is also closed under Shelah and Stupp's iteration (but not under Muchnik's iteration).

▶ **Theorem 13.** *For any $n > 0$, the set F_n is closed under basic iteration.*

To recognize languages, we fix an *input label* $\iota \in T$ and an *output label* $o \in T$. An *automaton* is a graph in which each input edge $s \xrightarrow{\iota} t$ and each output edge $s \xrightarrow{o} t$ is a loop: $s = t$. We denote by A the family of automata and $\mathrm{A}_n = \mathrm{A} \cap \mathrm{F}_n$ is the family of *level n automata* for any $n \geq 0$. We also consider the restriction $\mathrm{A}^{\mathrm{det}}$ of deterministic automata which have a deterministic graph with a unique loop labelled by ι. Any automaton G recognizes the language $\mathrm{L}(G)$ of path labels over $T_G - \{\iota, o\}$ from ι to o *i.e.*

$$\mathrm{L}(G) \;=\; \{\; u \in (T_G - \{\iota, o\})^* \mid \exists\, s, t \;(s \xrightarrow{\iota}_G s \xRightarrow{u}_G t \xrightarrow{o}_G t)\;\}.$$

For each $n \geq 0$, the n-indexed languages are the languages recognized by level n automata [6] ; we denote by Index_n this family:

$$\mathrm{Index}_n \;=\; \{\; \mathrm{L}(G) \mid G \in \mathrm{A}_n \;\}.$$

We also define the subfamily $\mathrm{Index}_n^{\mathrm{det}}$ of *n-indexed deterministic languages*:

$$\mathrm{Index}_n^{\mathrm{det}} \;=\; \{\; \mathrm{L}(G) \mid G \in \mathrm{F}_n \cap \mathrm{A}^{\mathrm{det}} \;\}.$$

So $\mathrm{Index}_0^{\mathrm{det}} = \mathrm{Index}_0$ is the family of regular languages, and $\mathrm{Index}_1^{\mathrm{det}}$ is the family of deterministic context-free languages. Recall that a *substitution* is a function $h : T \longrightarrow 2^{T^*}$ of finite domain that we extend by morphism: $h(uv) = h(u).h(v)$ for any $u, v \in (\mathrm{Dom}(h))^*$; we say that h is an Index_n-substitution for $n \geq 0$ if $h(a) \in \mathrm{Index}_n$ for all $a \in \mathrm{Dom}(h)$. The inverse substitution h^{-1} of a language $L \subseteq T^*$ is the language

$$h^{-1}(L) \;=\; \{\; u \in (\mathrm{Dom}(h))^* \mid h(u) \cap L \neq \emptyset \;\}.$$

An Index_0-substitution is a *regular substitution* which is a particular path function. Let us apply the closure of each family F_n under inverse path functions.

▶ **Corollary 14.** *For any $n \geq 0$, Index_n is closed under inverse regular substitutions.*

By Theorem 13, each family F_n is closed under synchronization product with finite automata.

▶ **Corollary 15.** *For any $n \geq 0$, the families Index_n and $\mathrm{Index}_n^{\mathrm{det}}$ are closed under intersection with any regular language.*

The Cayley graph $[R]$ of any Thue system R is extended to the *Cayley automaton* $[R, L]$ for any final set $L \subseteq \mathrm{Irr}_R$ by

$$[R, L] \;=\; [R] \;\cup\; \{\varepsilon \xrightarrow{\iota} \varepsilon\} \;\cup\; \{\; u \xrightarrow{o} u \mid u \in L \;\}.$$

where ι (resp. o) labelled loops mark initial (resp. final) states. For R canonical and by Lemma 2, $[R, L]$ is a deterministic and complete automaton recognizing by Proposition 4 the language $\mathrm{L}([R, L]) \;=\; \mathrm{Pre}_R^*(L) \;=\; [L]_{\leftrightarrow_R^*}$. Let us generalize Proposition 7.

▶ **Proposition 16.** *For any recognizable suffix system R, any $n \geq 0$ and $L \subseteq \mathrm{Irr}_R$ with $L \in \mathrm{Index}_n^{\mathrm{det}}$, we have $[R, L] \in \mathrm{F}_{n+1}$.*

This entails a generalization of Corollary 8 : Pre_R^* modifies by adding at most 1 the level of n-indexed deterministic languages when R is a confluent regular monadic system.

▶ **Theorem 17.** *For any recognizable system R which is canonical and suffix, for any language $L \subseteq \mathrm{Irr}_R$ and any $n \geq 0$, $L \in \mathrm{Index}_n^{\mathrm{det}} \implies \mathrm{Pre}_R^*(L) \;=\; [L]_{\leftrightarrow_R^*} \in \mathrm{Index}_{n+1}^{\mathrm{det}}$.*

Let us generalize Proposition 12 to indexed monadic systems. Like for the previous finite monadic system R_0, the rewriting $\to_R$ of an n-indexed monadic system R has in general an undecidable monadic theory, hence is not in the class F_n for any n. But for any n-indexed language L, we can recognize the language $\mathrm{Pre}^*_R(L)$ by a graph in F_n (in F_1 for $n = 0$). The construction uses *automaton substitutions* which are functions h of finite domain $\mathrm{Dom}(h) \subset T$ such that $h(a)$ is an automaton for each $a \in \mathrm{Dom}(h)$; we say that h is an F_n-substitution for some $n \geq 0$ if $h(a) \in \mathrm{F}_n$ for each $a \in \mathrm{Dom}(h)$. We also use *ε-automata* G allowing the label ε ($\varepsilon \in T_G$); its *ε-closure* is the automaton

$$G^\varepsilon \;=\; \{\; s \xrightarrow{a} t \mid s \overset{\varepsilon^*}{\Longrightarrow}_G \xrightarrow{a}_G \overset{\varepsilon^*}{\Longrightarrow}_G \;\wedge\; a \neq \varepsilon \;\} \;=\; g^{-1}(G)$$

for the path function g defined for any $a \in T_G - \{\varepsilon\}$ by $g(a) = \varepsilon^* a\, \varepsilon^*$. The image $h(G)$ of an automaton G by an automaton substitution h is the automaton

$$h(G) \;=\; (h_\varepsilon(G))^\varepsilon \;\cup\; \{\; s \xrightarrow{\iota} s \mid s \xrightarrow{\iota}_G s \;\} \;\cup\; \{\; s \xrightarrow{o} s \mid s \xrightarrow{o}_G s \;\}$$

where $h_\varepsilon(G)$ is the following ε-automaton:

$$\begin{array}{rcl} h_\varepsilon(G) & = & \bigcup_{(s,a,t)\in G} \{\; (s,a,p) \xrightarrow{b} (s,a,q) \mid p \xrightarrow{b}_{h(a)} q \;\wedge\; b \neq \iota \;\wedge\; b \neq o \;\} \\ & \cup & \{\; s \xrightarrow{\varepsilon} (s,a,q) \mid \exists\, t\; (s \xrightarrow{a}_G t) \;\wedge\; q \xrightarrow{\iota}_{h(a)} q \;\} \\ & \cup & \{\; (s,a,q) \xrightarrow{\varepsilon} t \mid s \xrightarrow{a}_G t \;\wedge\; q \xrightarrow{o}_{h(a)} q \;\}. \end{array}$$

To express the language recognized by $h(G)$, we associate to h the (language) substitution $\widehat{h}$ defined by $\widehat{h}(a) = \mathrm{L}(h(a))$ for any $a \in \mathrm{Dom}(h)$.

▶ **Lemma 18.** *For any automaton substitution h and any automaton G,*

$$\mathrm{L}(h(G)) \;=\; \widehat{h}(\mathrm{L}(G))$$

and for any $n \geq 0$, the automaton

$$h(G) \in \mathrm{F}_n \;\; \textit{for} \;\; G \in \mathrm{F}_n \;\; \textit{and} \;\; h \;\; \textit{is an} \;\; \mathrm{F}_n\textit{-substitution.}$$

Let us apply Lemma 18.

▶ **Corollary 19.** *For all $n \geq 0$, Index_n is closed under any Index_n-substitution.*

The *iterated automaton substitution* $h^*(G)$ of an automaton G by an automaton substitution h is the automaton

$$h^*(G) \;=\; \left(\textstyle\bigcup_{n\geq 0} h^n_\varepsilon(G)\right)^\varepsilon .$$

Similarly the *iterated language substitution* h^* of a (language) substitution h is the substitution of domain $\mathrm{Dom}(h)$ where the vector of languages $h^*(a)$ for $a \in \mathrm{Dom}(h)$ is the least fixed point of the system of equations

$$h^*(a) \;=\; \{a\} \cup h^*(h(a))$$

For $h(a) = aa$, we have $h^*(a) = a^+ \neq \bigcup_{n\geq 0} h^n(a) = \{\, a^{2^n} \mid n \geq 0 \,\}$. For the substitution h defined by $h(a) = bab$ and $h(b) = b$, we have $h^*(a) = \{\, b^n a b^n \mid n \geq 0 \,\}$ and $h^*(b) = b$. When h is a finite substitution, the equations defining h^* form a context-free grammar, hence h^* is a context-free substitution. Note that h^* remains a context-free substitution when h is a context-free substitution. To any automaton substitution h, we associate the monadic system $\overrightarrow{h} = \{\, (u,a) \mid a \in \mathrm{Dom}(h) \;\wedge\; u \in \mathrm{L}(h(a)) \,\}$. Let us iterate Lemma 18.

▶ **Lemma 20.** *For any automaton substitution h and any automaton G over $T_G \subseteq \mathrm{Dom}(h)$,*

$$\mathrm{L}(h^*(G)) \;=\; \widehat{h}^*(\mathrm{L}(G)) \;=\; \mathrm{Pre}^*_{\overrightarrow{h}}(\mathrm{L}(G))$$

and for any $n > 0$, the automaton

$$h^*(G) \in \mathrm{F}_n \;\; \textit{for} \;\; G \in \mathrm{F}_n \;\; \textit{and} \;\; h \;\; \textit{is an} \;\; \mathrm{F}_n\textit{-substitution.}$$

Let us apply Lemma 20.

▶ **Corollary 21.** *For all $n > 0$, any iterated Index_n-substitution is an Index_n-substitution.*

It remains to combine Theorem 13 with Lemma 20 to get for $n \neq 0$ that any n-indexed monadic system preserves n-indexed languages by inverse derivation.

▶ **Theorem 22.** *For any level $n \geq 1$ indexed monadic system R,*

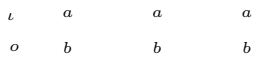
$$L \in \mathrm{Index}_n \;\; \Longrightarrow \;\; \mathrm{Pre}^*_R(L) \in \mathrm{Index}_n .$$

Let us combine Theorems 17 and 22.

▶ **Corollary 23.** *For any confluent regular monadic system R and any $n \geq 1$,*

$$L \;\in\; 2^{\mathrm{Irr}_R} \cap \mathrm{Index}_n^{\mathrm{det}} \;\; \Longrightarrow \;\; \mathrm{Pre}^*_R(L) \;\in\; \mathrm{Index}_n \cap \mathrm{Index}_{n+1}^{\mathrm{det}} .$$

For instance taking the finite system $R = \{(abc, b)\}$ which is monadic and confluent and taking the restricted Dyck language L over the pair (a, b) *i.e.* the language recognized by the automaton



which is an irreducible deterministic context-free language, the set

$$\mathrm{Pre}^*_R(L) \;=\; \{\; a^m a^{n_1} b c^{n_1} \ldots a^{n_m} b c^{n_m} \mid m \geq 0 \;\wedge\; n_1, \ldots, n_m \geq 0 \;\}$$

is a context-free language which is deterministic at level 2 but not at level 1.

5 Conclusion

We have generalized language preservation properties of regular and context-free monadic systems to higher-order indexed monadic systems. These results were obtained by applying two basic graph transformations: the basic iteration and inverse path functions.By applying Theorem 13 and Theorem 22 to the decomposition of the derivation of word rewriting systems [10], we can extend the preservation of context-free languages to n-indexed languages for each $n > 0$.

Acknowledgements Many thanks to Antoine Meyer for helping us make this paper readable, and to anonymous referees for helpful comments.

References

1 A. Aho, R. Sethi and J. Ullman, *Indexed grammars – an extension of context-free grammars*, JACM 15-4, 647–671 (1968).

2 M. Benois, *Parties rationnelles du groupe libre*, C.R. Académie des Sciences, Paris, Série A, 1188–1190 (1969).

3 R. Book, M. Jantzen and C. Wrathall, *Monadic Thue systems*, Theoretical Computer Science 19, 231–251 (1982).

4 R. Book and F. Otto, *String-rewriting systems*, Texts and Monographs in Computer Science, Springer-Verlag, 189 pages (1993).

5 H. Calbrix and T. Knapik, *A string-rewriting characterization of Muller and Schupp's context-free graphs*, 18^{th} FSTTCS, LNCS 1530, V. Arvind, R. Ramanujam (Eds.), 331–342 (1998).

6 A. Carayol and S. Wöhrle, *The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata*, 23^{rd} FSTTCS, LNCS 2914, P. Pandya, J. Radhakrishnan (Eds.), 112–123 (2003).

7 D. Caucal, *On infinite transition graphs having a decidable monadic theory*, 23^{rd} ICALP, LNCS 1099, F. Meyer auf der Heide, B. Monien (Eds.), 194–205 (1996)
or in Theoretical Computer Science 290, 79–115 (2003).

8 D. Caucal, *On the transition graphs of Turing machines*, 3^{rd} MCU, LNCS 2055, M. Margenstern, Y. Rogozhin (Eds.), 177–189 (2001).

9 D. Caucal, *On infinite terms having a decidable monadic theory*, 27^{th} MFCS, LNCS 2420, K. Diks, W. Rytter (Eds.), 165–176 (2002).

10 D. Caucal and T.H. Dinh, *Regularity and context-freeness over word rewriting systems*, 14^{th} FOSSACS, LNCS 6604, Martin Hofmann (Ed.), 214–228 (2011).

11 J. Engelfriet and E. Schmidt, *IO and OI*, Journal of Computer and System Sciences 15, 328–353 (1977).

12 M. Jantzen, M. Kudlek, K.-J. Lange and H. Petersen, *$Dyck_1$-reductions of context-free languages*, 6^{th} FCT, LNCS 278, L. Budach, R. Bakharajev, O. Lipanov (Eds.), 218–227 (1987).

13 A. Maslov, *The hierarchy of indexed languages of arbitrary level*, Doklady Akademii Nauk SSSR 217, 1013–1016 (1974).

14 A. Maslov, *Multilevel pushdown automata*, Problemy Peredacy Informacii 12-1, 55–62 (1976).

15 C. Ó'Dúnlaing, *Infinite regular Thue systems*, Theoretical Computer Science 25, 171–192 (1983).

16 A. Semenov, *Decidability of monadic theories*, 11^{th} MFCS, LNCS 176, M. Chytil, V. Koubek (Eds.), 162–175 (1984).

17 S. Shelah, *The monadic theory of order*, Annals of Mathematics 102, 379–419 (1975).

18 J. Stupp, *The lattice model is recursive in the original model*, The Hebrew University (1975).

18 I. Walukiewicz, *Monadic second-order logic on tree-like structures*, Theoretical Computer Science 275, 311–346 (2002).

19 S. Wöhrle and W. Thomas, *Model checking synchronized products of infinite transition systems*, Logical Methods in Computer Science 3 (4:5), 1–18 (2007).

20 S. Yu, *A pumping lemma for deterministic context-free languages*, Information Processing Letters 31-1, 47–51 (1989).